Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterthesis

Moritz Verbeck

Smart Heat Grid Hamburg: Konzeptionierung und prototypische Umsetzung einer intelligenten Wärmeübergabestation zur Erprobung einer angepassten SPS-Programmiermethodik

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Moritz Verbeck

Smart Heat Grid Hamburg: Konzeptionierung und prototypische Umsetzung einer intelligenten Wärmeübergabestation zur Erprobung einer angepassten SPS-Programmiermethodik

Masterthesis eingereicht im Rahmen der Masterprüfung
im Masterstudiengang Automatisierung
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Franz Schubert
Zweitgutachter : Prof. Dr. Hans Schäfers

Abgegeben am 6. Mai 2019

Moritz Verbeck

Thema der Masterthesis

Smart Heat Grid Hamburg: Konzeptionierung und prototypische Umsetzung einer intelligenten Wärmeübergabestation zur Erprobung einer angepassten SPS-Programmiermethodik

Stichworte

iWÜST, intelligente Wärmeübergabestation, Parser, SPS, Regelalgorithmus, Strukturierter Text, Smart heat grid, Modellierung einer Wärmeübergabestation

Kurzzusammenfassung

In dieser Arbeit wird ein Regelalgorithmus für eine intelligente Wärmeübergabestation umgesetzt und anhand einer speziellen Methodik validiert. Dafür wird ein Tool programmiert und in die Simulationsumgebung *Jarvis* integriert, welches Informationen eines Steuerungsprogramms der Entwicklungsumgebung *CoDeSys 2.3* extrahiert und Steuerungscode der Programmiersprache *Strukturierter Text* in *Python* übersetzt. Für die Validierung wird eine SPS-basierte Hardwaretestumgebung sowie ein Softwaremodell in *Jarvis* erstellt. Anhand dieser Modelle werden anschließend die verschiedenen Regelszenarien getestet und bewertet.

Moritz Verbeck

Title of the paper

Smart Heat Grid Hamburg: Conceptual design and prototypical implementation of an intelligent district heating substation for testing an adapted PLC programming methodology

Keywords

iWÜST, intelligent district heating substation, parser, PLC, control algorithm, Structured text, smart heat grid, modelling of a district heating substation

Abstract

In this thesis, a control algorithm for an intelligent district heating substation is implemented and validated using a specific methodology. Therefor a tool is programmed and integrated into the simulation environment *Jarvis*. The tool extracts information from a control programm of the control system software *CoDeSys 2.3* and translates control code of the programming language *Structured text* into *Python*. For validation, a PLC-based hardware test bed and a software model in *Jarvis* are created. Various control scenarios are then tested and evaluated on the base of these models.

Danksagung

An dieser Stelle möchte ich mich bei allen MitarbeiterInnen des SHGH-Teams für die fachliche und persönliche Unterstützung während der Anfertigung dieser Arbeit bedanken. Mein besonderer Dank gilt dabei Philipp, Peter und Fabian für die Betreuung und das Korrekturlesen.

Herrn Prof. Franz Schubert und Herrn Prof. Hans Schäfers danke ich für die Übernahme des Erst- und Zweitgutachtens.

Zu guter Letzt möchte ich vor allem meiner Familie, meinen Freunden Igor, Johannes, Richard, Anne, Eneko und Michel sowie meinen Mitbewohnern Laura, Manuel, Benjamin, Lukas und allen weiteren, nicht einzeln genannten lieben Menschen aus meinem Umfeld dafür danken, dass sie stets ein offenens Ohr für meine Probleme haben und immer hinter mir stehen.

Vielen Dank!

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungs- und Formelverzeichnis

Abkürzung	**Bedeutung**
SHGH	Smart Heat Grid Hamburg
BMWi	Bundesministerium für Wirtschaft und Energie
WÜST	Wärmeübergabestation
iWÜST	intelligente Wärmeübergabestation
TWW	Trinkwarmwasserversorgung
ST	Strukturierter Text
IKT	Informations- und Kommunikations-Technologie
BHKW	Blockheizkraftwerk
SPS	Speicherprogrammierbare Steuerung
bspw.	beispielsweise
z.B.	zum Beispiel
SOC	Speicherladezustand
R&I	Rohrleitungs- und Instrumentenfließschema
FUP	Funktionsplan
CPU	Recheneinheit
GUI	Grafische Benutzeroberfläche
PID	Proportional-Integral-Derivative
Sz.-Nr.	Szenarionummer
DIN	Deutsches Institut für Normung e.V.
LED	Leuchtdiode

Symbol	**Bedeutung**
A	Fläche des Wärmeübertragers
c_p	Wärmekapazität von Wasser
$\dot{Q}$	Wärmestrom
k	Wärmedurchgangskoeffizient
T	Temperatur
ΔT_m	mittlere Temperaturdifferenz
Δt_{max}	maximale Temperaturdifferenz
Δt_{min}	minimale Temperaturdifferenz
$\dot{m}$	Massenstrom
T_{med}	arithmetrisches Mittel der Temperaturen
T_RL_{min}	minimale Rücklauftemperatur
T_Netz_{mittel}	gemittelte Netztemperatur
t	Zeit
KP	Verstärkungsfaktor
TN	Nachstellzeit
TV	Vorhaltezeit

1. Einleitung

In diesem Kapitel wird auf die Motivation, das Projekt *Smart Heat Grid Hamburg*, die Aufgabenstellung sowie den Aufbau dieser Masterarbeit eingegangen.

1.1. Motivation

Auf der UN-Klimakonferenz 2015 in Paris wurde zum ersten Mal von allen Ländern der Welt ein Abkommen beschlossen, welches die Erderwärmung auf weniger als $2\,^{\circ}\mathrm{C}$ begrenzen soll. Des Weiteren einigten sich die Experten darauf, dass ab der zweiten Hälfte des Jahrhunderts eine weltweite Treibhausgasneutralität herrschen muss [1]. Zur Dokumentation der Anstrengungen und Ziele der einzelnen Staaten wurden auf der Konferenz 2018 in Katowice Normen und Richtlinien bestimmt. Die von Deutschland gesetzten Ziele sollen durch die im Jahre 2010 ausgerufene Energiewende erreicht werden. Der ein Jahr später beschlossene Ausstieg aus der Atomenergie sowie der aktuell durch die Kohlekommission behandelte Ausstieg aus der Braunkohle [2] stellt eine Herkulesaufgabe für die Energieversorgung dar. Die noch für die Stromerzeugung genutzten fossilen Kraftwerke sollen in relativ kurzer Zeit suksessiv durch Anlagen zur Erzeugung von erneuerbarer, klimaschonender Energie ersetzt werden. Im Verkehrssektor soll der Ausstoß von Treibhausgasen vor allem durch eine Erhöhung des Anteils an Elektrofahrzeugen vermindert werden. Der Wärmesektor, welcher deutschlandweit $56\,\%$ des gesamten Endenergieverbrauchs ausmacht [3], soll ebenfalls durch weitreichende Änderungen nachhaltiger gemacht werden.

Aufgrund der Wetterabhängigkeit von regenerativen Energiequellen werden für diesen Paradigmenwechsel in der Energieversorgung Systeme benötigt, die die Fluktuation der Erzeugung ausgleichen. Eine Lösung besteht in der Kopplung der Sektoren Strom, Wärme und Verkehr. Die intelligente Vernetzung unterschiedlicher Energiesysteme ermöglicht die Reduktion von CO_2-Emissionen durch einen effizienteren Einsatz der vorhandenen Energie sowie der Substitution von fossilen Energiequellen. Beispielsweise können Strom- und Wärmesektor gekoppelt werden, indem überschüssiger, erneuerbarer Strom von volatilen Erzeugern durch die Nutzung von Wärmepumpen oder Power-to-heat-Anlagen für die Wärmebereitstellung genutzt werden. Dabei werden die aktuellen und zukünftig prognostizierten Verbräuche im Strom- und Wärmesektor abgeglichen, sodass der Einsatz unterschiedlicher

Erzeugungsanlagen unter ökologischen und ökonomischen Gesichtspunkten abgestimmt werden kann. Des Weiteren können Wetter- und Verbrauchsprognosen miteinbezogen werden, sodass Lade- und Entladestrategien für die elektrischen und thermischen Speicher mittels Algorithmen berechnet werden können. Die Speicherung der Energie ermöglicht die für die Energiewende notwendige Entkopplung von Energieerzeugung und Energienachfrage. Für diese weitreichenden Neuerungen im Feld der Energieerzeugung, der Übertragung und des Verbrauchs müssen Konzepte für das Erreichen der Ziele gefunden und entwickelt werden.

Im Rahmen des Projektes “Smart Heat Grid Hamburg"(SHGH), das Teil des Förderprogramms “EnEff:Wärme"des Bundesministeriums für Wirtschaft und Energie (BMWi) ist, sollen für die oben genannten Herausforderungen Lösungen für Wärmenetze erarbeitet werden. Dabei werden die Wärmenetze unter anderem durch die Einbindung der Kundenseite weiterentwickelt. Durch eine integrierte Steuerung der Wärmeübergabestation kann die Abnahme der Wärme gesteuert werden, wodurch eine netzdienliche, effiziente und sichere Versorgung der Kunden möglich ist.

1.2. Smart Heat Grid Hamburg

Aufbauend auf die Erfahrungen und Ergebnisse des Forschungsprojektes “Smart Power Hamburg”, in dem eine offene IKT-Energieplattform entwickelt wurde, ist das Projekt SHGH entstanden. Neben dem Abrufen aktueller Messdaten der im Verbund vorhandenen Anlagen können mit Hilfe einer Energieplattform auch Aktoren angesteuert werden, sodass bspw. im Anwendungsfall eines Wärmenetzes Temperaturen und Drücke erfasst und die Pumpendrehzahl oder Leistung eines Blockheizkraftwerks (BHKW) extern gezielt angepasst werden können [4]. “Die beteiligten Akteure des Projektes Smart Heat Grid Hamburg (Hochschule für angewandte Wissenschaften Hamburg, HAMBURG ENERGIE GmbH und eNeG Gesellschaft für wirtschaftlichen Energieeinsatz mbH) haben es sich zur Aufgabe gemacht, innerhalb der vierjährigen Projektlaufzeit Konzepte zu entwickeln, um den Anteil der Erneuerbaren Energien sowie die Energieeffizienz im Wärmenetz durch die Einbindung von intelligenter Wärmeinfrastruktur zu maximieren."[5]

Zu diesem Zweck soll u.a. die Entwicklung einer Simulationsumgebung zur Modellierung und Simulation komplexer Wärmenetzstrukturen und deren Auslegung an Hand relevanter physikalischer Parameter erfolgen. Dadurch sollen verschiedene Betriebskonzepte hinsichtlich ihrer Umweltverträglichkeit und Wirtschaftlichkeit analysiert und angepasst werden können. Die dabei gewonnenen Erkenntnisse sollen anschließend im Wärmenetz Wilhelmsburg von HAMBURG ENERGIE getestet und angepasst werden. Ein innovativer Bestandteil des Wärmenetzes ist der 2013 fertiggestellte Energiebunker, welcher als regeneratives Wärmekraftwerk mit Großwärmespeicher dient [6]. Es sollen außerdem im Zuge

des Projektes Mess-, Steuer- und Regelungskonzepte standardisiert und anschließend im Feld getestet werden. Zur Steigerung der Flexibilität des Netzes werden kundennahe Wärmespeicher mit intelligenter Steuerung und angepasster Versorgungsgrenze entwickelt und getestet sowie eine aktive Integration der Sekundärseite in Betracht gezogen. Betriebsüberwachungskonzepte werden entwickelt, um in Echtzeit den korrekten Ablauf der geplanten Prozesse zu überprüfen [5].

1.3. Aufgabenstellung

Die Aufgabenstellung dieser Arbeit lässt sich in drei Bereiche unterteilen. Im ersten Schritt soll ein Programm entwickelt werden, welches bestimmte Informationen aus Exportdateien der Entwicklungsumgebung *CoDeSys 2.3* für die Weiterverwendung in der Simulationsumgebung herausfiltert. Zusätzlich soll Code der Programmiersprache *Strukturierter Text* (ST) der internationalen Norm IEC61131-3 aus der Exportdatei in die Programmiersprache *Python* gewandelt werden. Anschließend soll eine Hardwaretestumgebung erstellt werden, die eine intelligente Wärmeübergabestation (iWÜST) inklusive der Sensoren, Aktoren und einer Speicherprogrammierbaren Steuerung (SPS) abbildet. Ein Regelalgorithmus zur Steuerung der Hardwaretestumgebung soll mittels *CoDeSys 2.3* für die SPS erstellt werden. Zur Validierung des Parsers, der Hardwaretestumgebung sowie des Regelalgorithmus wird die iWÜST in der Simulationsumgebung *Jarvis* modelliert und unter Verwendung des Regelalgorithmus getestet.

1.4. Aufbau

Neben einer kurzen Einführung in die Fernwärme wird eine herkömmliche Wärmeübergabestation kurz erklärt und die Besonderheiten sowie der Mehrwert einer iWÜST im Bezug dazu dargelegt. Anschließend werden die Begrifflichkeiten Parser, *Jarvis* und SPS erläutert. Im dritten Kapitel wird die Entwicklung des Parsers beschrieben, wie dieser in Jarvis eingebunden ist und verwendet werden kann. Kapitel 4 widmet sich der Umsetzung des Hardwaremodells. Im Kapitel 5 wird das in der Simulationsumgebung *Jarvis* erstellte Modell der iWÜST erklärt. Im Anschluss folgt die Realisierung des Regelalgorithmus unter Berücksichtigung der Regelstrategien und -szenarien. Anhand der Ergebnisse aus Simulation und Hardwaretestumgebung wird im letzten Kapitel ein Resümee bezüglich der entwickelten Steuerungsalgorithmik und des Parsers gezogen. Im Rahmen eines Ausblickes werden mögliche weiterführende Entwicklungsmöglichkeiten gezeigt.

2. Grundlagen

2.1. Wärmenetze

Deutschlandweit wurden im Jahr 2017 ca. 14 % der Haushalte mittels Fernwärmeanschluss versorgt [7]. In Fernwärmenetzen wird im wesentlichen Wasser an zentralen Orten erhitzt und über Rohrleitungssysteme an die Verbraucher geleitet. Diese nehmen die Wärme auf und nutzen sie zum Heizen von Räumen, zum Erhitzen von Trinkwarmwasser oder für industrielle Prozesse. Das dadurch abgekühlte Wasser wird wiederum durch Leitungen an die Wärmeerzeuger zurückgeführt, sodass der Kreislauf geschlossen ist. Dies ist schematisch in Abbildung 2.1 gezeigt. Diese Darstellung enthält zwischen Erzeugeranlage und Pumpstation einen Speicher zum Ausgleich von Netzschwankungen, welcher jedoch nicht unbedingt benötigt wird. Aktuell wird die Wärme in Deutschland größtenteils über Kraft-Wärme-Kopplungsanlagen

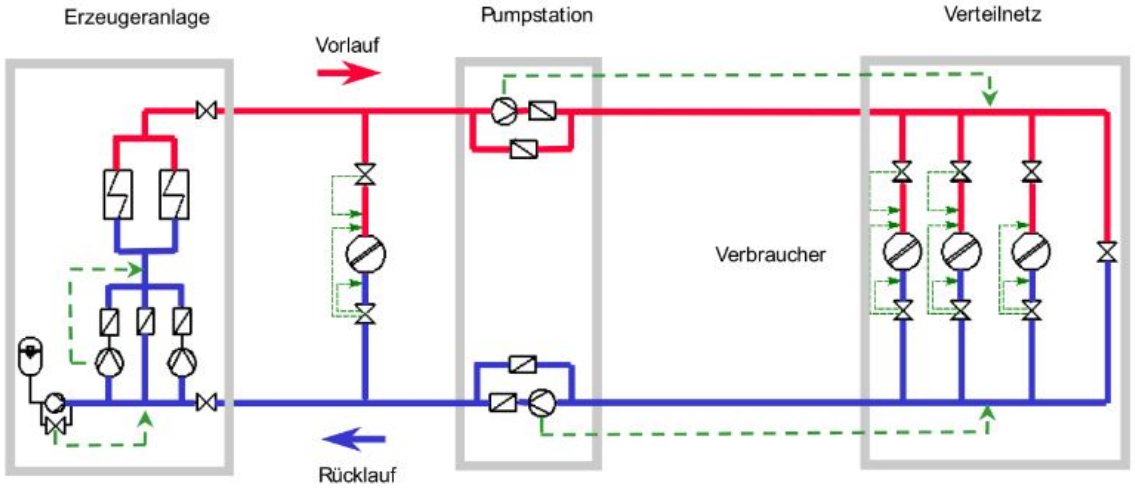


Abbildung 2.1.: Modell eines Fernwärmesystems [8]

erzeugt [9]. Abbildung 2.2 zeigt den historischen Entwicklungsprozess der Wärmenetze. Es ist zu erkennen, dass bei den zukünftigen Netzen der Generation 4.0 die Möglichkeit besteht, dass eine Vielzahl von Erzeugern für die Bereitstellung von Wärme zuständig ist. Die gezeigte Entwicklung des Temperaturniveaus macht deutlich, dass die im Vorlauf des

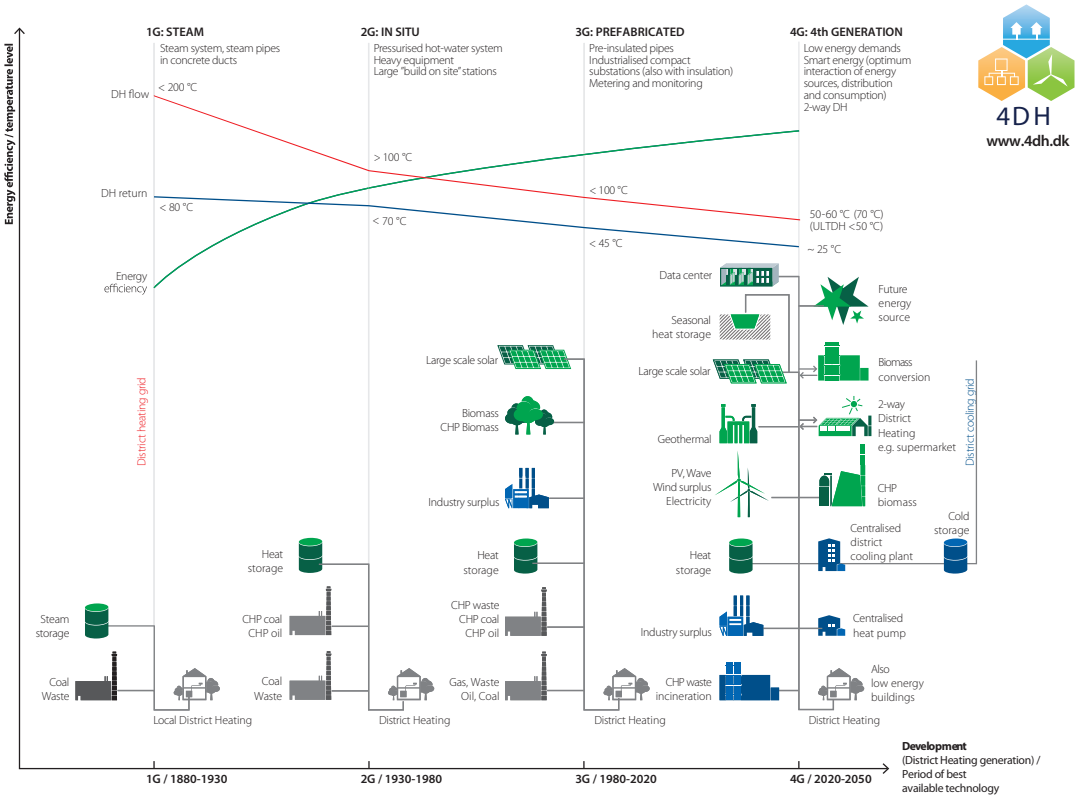


Abbildung 2.2.: Historische Entwicklung der Fernwärme [10]

Netzes benötigte Wassertemperatur fällt, wodurch die Energieeffizienz steigt. Dadurch wird außerdem ermöglicht, dass Anlagen mit niedrigeren Prozesstemperaturen ebenfalls Wärme in das Netz speisen können. Beispielsweise kann die Umweltverträglichkeit durch Einbezug von industrieller Abwärme oder Wandlung von überschüssigem, erneuerbar erzeugtem Strom mittels Power-to-Heat-Anlagen oder Wärmepumpen erhöht werden. Geothermiekraftwerke können in großem Maße Wärme bereitstellen. Fluktuationen können in diesen Systemen durch große, saisonale sowie durch kleine Wärmespeicher ausgeglichen werden. Die Anpassung der vorhandenen Fernwärmestrukturen bedarf unter anderem einer weitreichenden Optimierung der Steuerungs- und Regelungstechnik. Durch die damit einhergehende, eventuelle nachträgliche Dezentralisierung der Einspeisung in das Fernwärmenetz ändern sich bspw. Netzschlechtpunkte im System. Weitere Temperatur- und Drucksensoren müssen vorgesehen und in einer Leitwarte analysiert werden. Durch die Kopplung mit dem Stromsektor werden

Speichermöglichkeiten sowie -szenarien benötigt, sodass auf äußere Faktoren wie z.B. das Wetter reagiert werden kann.

2.2. Wärmeübergabestation

Wärmeübergabestationen (WÜST) sind Anlagen, die thermische Energie aus einer Fernwärmeleitung in Wärmeverteilsysteme (Heizkreise) von Kunden übertragen. Sie sind somit das Bindeglied zwischen Wärmenetz und Hauszentrale. Man unterscheidet bei der Art des Anschlusses zwischen direkt und indirekt betriebenen Systemen. Bei Ersteren wird die Hausanlage mit dem Wärmeträgermedium aus der Fernwärmeleitung durchströmt. In dem Fall muss das Haussystem auf die Drücke, Temperaturen und die Wasserqualität des Fernwärmenetzes ausgelegt werden. Bei indirekt betriebenen Systemen ist der Wasserkreislauf des Wärmenetzes von der sekundärseitigen Kundenanlage durch einen Wärmeübertrager hydraulisch getrennt. Diese Systeme werden zunehmend favorisiert, da dadurch eine höhere Flexibilität bezüglich der Netztemperatur und der Temperaturen in der Hausanlage ermöglicht wird. Außerdem werden nicht so leistungsstarke Netzpumpen wie bei den direkt angeschlossenen Systemen benötigt. Im betrachteten Wärmenetz sind ausschließlich indirekte Anschlussstationen verbaut. Daher bezieht sich der weitere Verlauf ausschließlich auf diese.

2.2.1. Bestandteile

Hauptbestandteil der indirekten WÜST ist der Wärmeübertrager. Die Leistung $\dot{Q}$, welche von einem bestimmten Wärmeübertrager übertragen werden kann, wird mittels Gleichung 2.1 definiert:

$$\dot{Q} = k \cdot A \cdot \Delta T_m \tag{2.1}$$

$$\Delta T_m = \frac{\Delta t_{max} - \Delta t_{min}}{ln(\frac{\Delta t_{max}}{\Delta t_{min}})} \tag{2.2}$$

Dabei ist k der Wärmedurchgangskoeffizient des Wärmeübertragers, A die Wärmeübertragungsfläche des Wärmeübertragers und ΔT_m die mittlere Temperaturdifferenz, welche mit Formel 2.2 berechnet werden kann [12]. Wärmeübertrager können als Platten, Rohrbündel oder Mantelrohr ausgeführt sein. Um das komplette Potential des Wärmeübertragers ausnutzen zu können, muss dieser eine ausreichende Fläche aufweisen und ordnungsgemäß

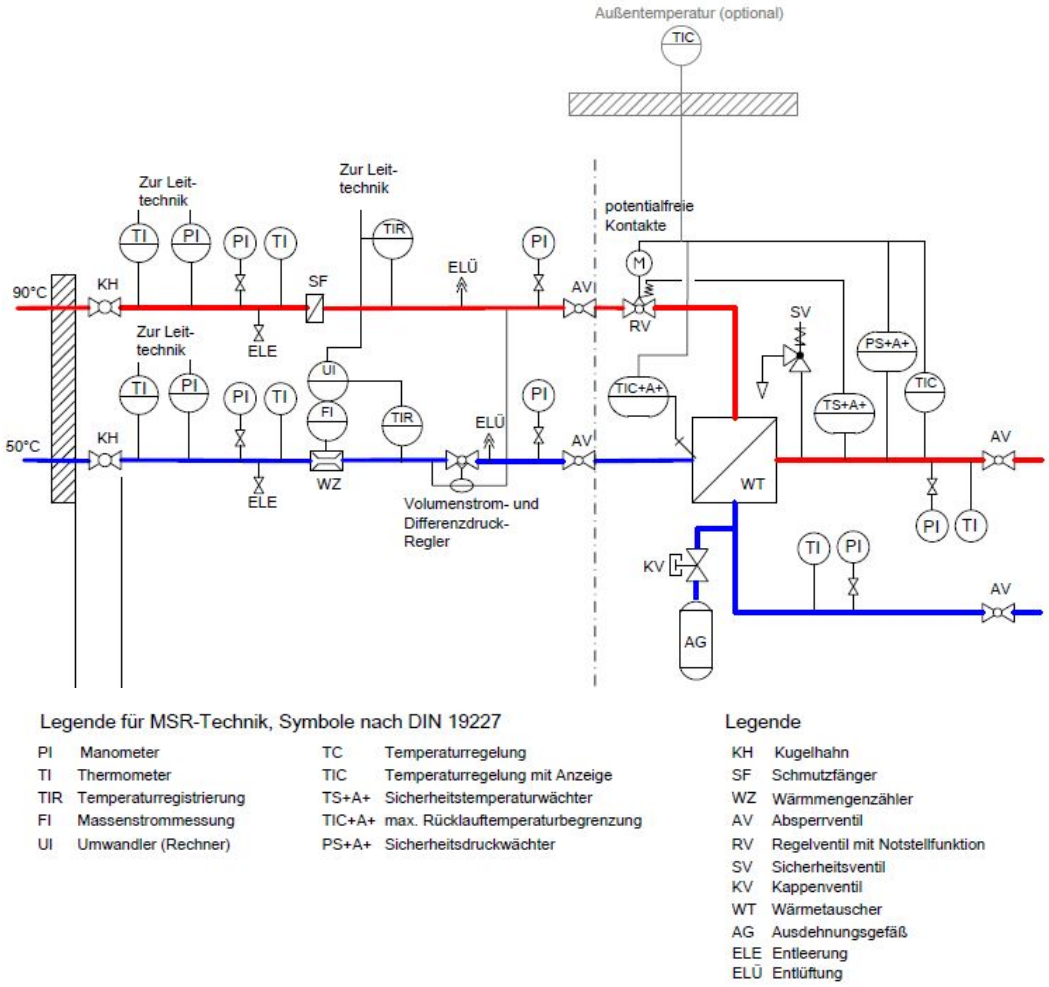


Abbildung 2.3.: Wärmeübergabestation ohne Trinkwarmwassererwärmung [11]

betrieben werden. Die Wärmemenge $\dot{Q}$, die zwischen zwei Anschlüssen dem System entnommen wird, kann mit der Gleichung 2.3 berechnet werden. c_p ist dabei die spezifische Wärmekapazität, $\dot{m}$ der Massenstrom und ΔT der Temperaturunterschied zwischen Vor- und Rücklauf.

$$\dot{Q} = c_p \cdot \dot{m} \cdot \Delta T \tag{2.3}$$

Als Wärmeträgermedium wird im Wärmenetz sowie auf der Sekundärseite üblicherweise aufbereitetes (entsalztes), flüssiges Wasser genutzt. Die Temperatur im Vorlauf liegt, abhängig vom Druckniveau des Netzes, üblicherweise zwischen 70 °C und 130 °C, wobei unter Umständen auch höhere Temperaturen und Dampfnetze möglich sind [8]. Neben den Anschlussleitungen und dem Wärmeübertrager enthalten die WÜST im betrachteten Wärmenetz

digital auslesbare Temperatur- und Drucksensoren, Messeinrichtungen wie Wärmemengenzähler und Aktoren (Stell- und Mischventile, Pumpen), welche an eine lokale Steuereinheit angeschlossen sind. Analoge Messinstrumente ermöglichen eine Überprüfung der Systemdrücke und -temperaturen vor Ort. Abbildung 2.3 zeigt exemplarisch den hydraulischen und messtechnischen Aufbau einer Wärmeübergabestation. In dem Schema sind die standardmäßig installierten Komponenten dargestellt. Dieser Aufbau kann um einen Anschluss für eine Trinkwarmwasserversorgung (TWW) erweitert werden. Dabei wird zu dem Heizkreis für die Raumheizung üblicherweise ein zweiter Wärmeübertrager parallel angeschlossen, sodass das Trinkwarmwasser auch mittels Durchflusserhitzung erwärmt wird. Optional können weitere Heizkreise eingebaut werden. Bei Gebäuden ohne angeschlossener TWW wird das Trinkwarmwasser meistens durch elektrische Durchlauferhitzer an den einzelnen Zapfstellen bereitgestellt.

2.2.2. Regelung

Ein motorbetriebenes Durchgangsventil im Rücklauf der Primärseite regelt für gewöhnlich den wärmenetzseitigen Volumenstrom so, dass sich eine bestimmte Vorlauftemperatur im Sekundärkreislauf einstellt (siehe Abbildung 2.4). In der Regel ist der Sollwert der Vorlauftemperatur abhängig von der Außentemperatur. Aus der Vorlauftemperatur, der Leistungsabnahme, dem Volumenstrom und den Dimensionen des Wärmeübertragers ergibt sich die primärseitige Rücklauftemperatur. Im Bezug auf das Wärmenetz soll sich bestensfalls eine möglichst niedrige primärseitige Rücklauftemperatur einstellen. Eine Variante der Regelstrategie ist die Integration einer Begrenzung der Rücklauftemperatur in den Regler. Dabei ändert sich die Regelung des Durchgangsventils im Rücklauf der Primärseite bei Überschreitung eines Grenzwertes vorübergehend, sodass nicht mehr die sekundäre Vorlauftemperatur, sondern die primäre Rücklauftemperatur den Sollwert vorgibt.

Die vorgegebenen Vorlauftemperaturen der Verbraucher in den einzelnen Wasserkreisläufen der Hausanlage werden mittels einer der in Abbildung 2.5 dargestellten Regelungen vorgenommen. In den Abbildungen entspricht der “Erzeuger” für die hier beschriebenen Regelungen einer WÜST. Neben Vorlauftemperatur der Hausanlage und maximaler Rücklauftemperatur kann bei den üblichen Steuerungen der WÜST als dritter Regelparameter eine vertraglich vereinbarte Anschlussleistung definiert werden, welche bei der Regelung berücksichtigt wird [13]. Die abgegebene Wärmeleistung kann mittels Temperaturdifferenz von primärseitigem Vor- und Rücklauf und Massenstrom berechnet werden und wird in der Praxis durch einen Wärmemengenzähler erfasst [14]. Diese Parameter sind vorrangig für die Einstellung des Ventils verantwortlich. Die Steuerung der Aktoren der WÜST wird im Regelfall von einer SPS übernommen, welche im Kapitel 2.3 erklärt werden.

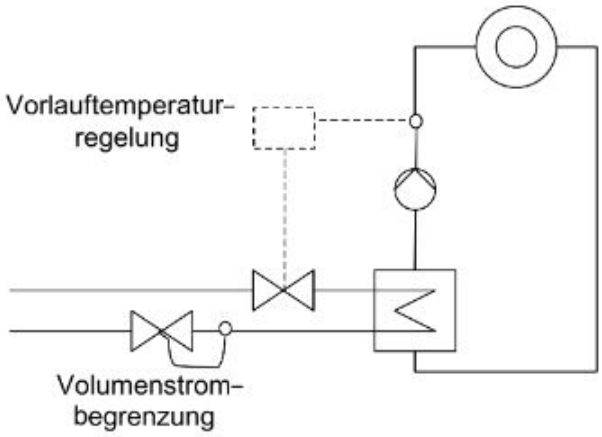


Abbildung 2.4.: Regelung der gebäudeseitigen Vorlauftemperatur mittels primärseitigem Durchgangsventil [8]

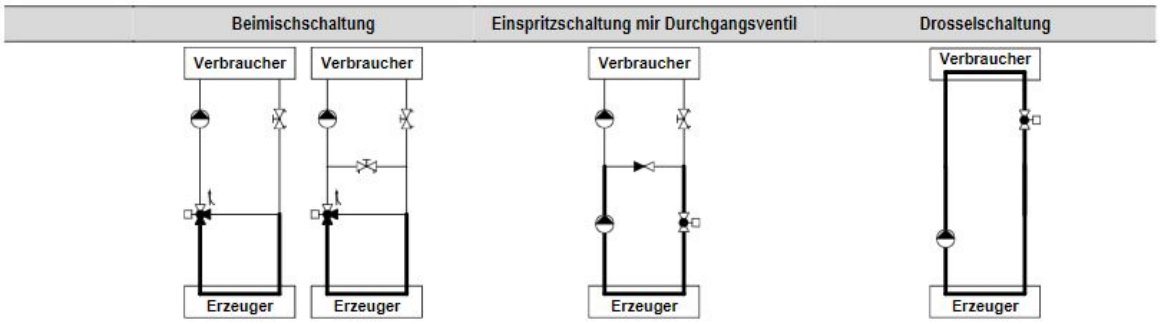


Abbildung 2.5.: Hydraulische Grundschaltungen in der Gebäudetechnik [13]

2.2.3. Intelligente Wärmeübergabestation

Bei der in dieser Arbeit behandelten iWÜST handelt es sich hydraulisch um eine WÜST, welche mittels Wärmeübertrager einen oder mehrere Heizkreise versorgt und eine dazu parallel angeschlossene TWW, die einen Wärmespeicher und eine WÜST umfasst. Die Besonderheit der iWÜST ist neben der hydraulischen Verschaltung des Wärmespeichers die Anzahl der erfassten Daten sowie die Möglichkeit, durch den Datenaustausch mit der Leitwarte auf verschiedene Netzsituationen reagieren zu können. Das Konzept sowie die erste prototypische Umsetzung der iWÜST wurde vom Projekt SHGH entwickelt. Ein Foto des Speichers mit der WÜST zur TWW ist in Abbildung 2.6 gezeigt. Diese im Wärmenetz integrierte Anlage wird in dieser Arbeit als “reale iWÜST” bezeichnet und dient häufig als Anhaltspunkt für Dimensionierungen und Parametrierungen.

Abbildung 2.6.: Wärmespeicher und WÜST der realen iWÜST [15]

Der Wärmeübertrager der TWW kann entweder direkt aus der Fernwärmeleitung oder von einem parallel angeschlossenen Wärmespeicher versorgt werden, welcher einen mittigen Einspeisestutzen besitzt. Die hydraulische Verschaltung der TWW inklusive des Mischventils und der Pumpe im TWW-Kreislauf ermöglicht verschiedene Lade- und Entladeszenarien des Speichers. Dadurch kann das Verhalten an die aktuelle und prognostizierte Situation im Wärmenetz angepasst und sodann auf Betriebsanforderungen reagiert werden. Netzengpässe oder hohe Einspeisung durch erneuerbar gewonnene Wärme können dadurch abgefedert werden. Das Rohrleitungs- und Instrumentenfließschema (R&I) der iWÜST ist in Abbildung 2.8 auf Seite 12 dargestellt.

Im Gegensatz zu konventionellen WÜST verschiebt sich die Eigentumsgrenze des Fernwärmenetzbetreibers und des Kundens. Üblicherweise endet dieser vor dem Wärmeübertrager, sodass dieser vom Kunden betrieben, geregelt und gewartet wird. Für eine größtmögliche Flexibilität und Stabilität im Netz ist es von Vorteil, wenn der Netzbetreiber Zugang zu möglichst vielen Datenpunkten und Aktoren hat. Dadurch ist eine Verschiebung der Liefergrenze in Richtung des Kundens von nöten. Im vereinfachten Schema der iWÜST mit TWW in Abbildung 2.7 ist die Grenze des Zuständigkeitsbereichs des Wärmenetzbetreibers zum Kunden zu erkennen. Der Netzbetreiber ist für die korrekte Funktionalität aller in Abbildung 2.7 gezeigten Komponenten verantwortlich.

Steuerungstechnisch besitzt die iWÜST neben einer Vielzahl von Temperatur- und Drucksensoren drei stellmotorbetriebene Ventile (zweimal Durchgang, ein Mischer), eine Pumpe und einen Wärmemengenzähler. Der Speicherladezustand (engl. State of charge, SOC) kann mit

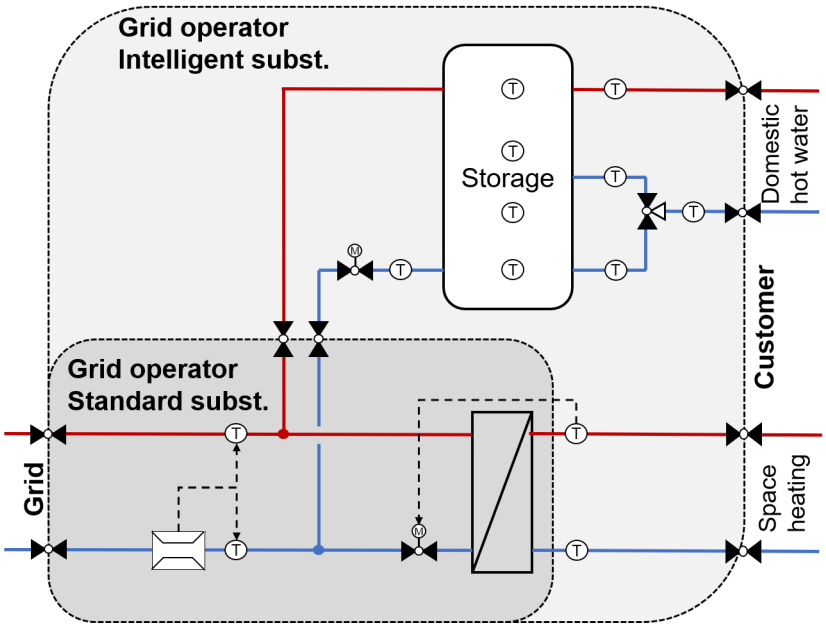


Abbildung 2.7.: Vereinfachtes Schema der realen iWÜST [15]

Hilfe der Temperatursensoren im Speicher berechnet werden. Wie bei der konventionellen WÜST sind die Aktoren und Sensoren an eine SPS angeschlossen, welche die Regelung der Ventile und der Pumpe übernimmt. Sie kommuniziert über Ethernet mit einer Leitwarte, die aktuelle Systemdaten der iWÜST empfängt und, bei Bedarf, Szenarien auslöst. Szenarien können außerdem aktiviert werden, indem bestimmte Messsignale vorgegebene Grenzwerte über- oder unterschreiten. Wurde ein Szenario ausgelöst, so werden bestimmte Sollwerte für die verschiedenen Regelgrößen vorgegeben, sodass die Regelung der iWÜST mit unterschiedlichen Ventil- und Pumpenstellungen auf die in den Tabellen 2.1 und 2.2 angegebenen, vom Netz- und Systemzustand abhängigen Szenarien reagiert. Die Szenarien sorgen dafür, dass eine möglichst netzdienliche, flexible und sichere Versorgung der Kunden mit Wärme sichergestellt ist.

Die Szenarien sind in den Regelbeschreibungen der iWÜST ohne TWW und der iWÜST mit TWW beschrieben. Diese wurden von einer Gruppe, bestehend aus Mitarbeitern der am Projekt *SHGH* beteiligten Projektpartnern, erstellt und sind in den Abbildungen A.3 und A.1 im Anhang zu finden. Die beiden Regelbeschreibungen unterscheiden sich hinsichtlich des zu regelnden Systems. Das eine Systeme enthält eine iWÜST mit parallel angeschlossenem Speicher (iWÜST mit TWW). Dadurch ergeben sich andere Regelungsmöglichkeiten als bei Systemen ohne Speicher (iWÜST ohne TWW). Bei Systemen ohne TWW werden nur die Szenarien aus Tabelle 2.2 berücksichtigt.

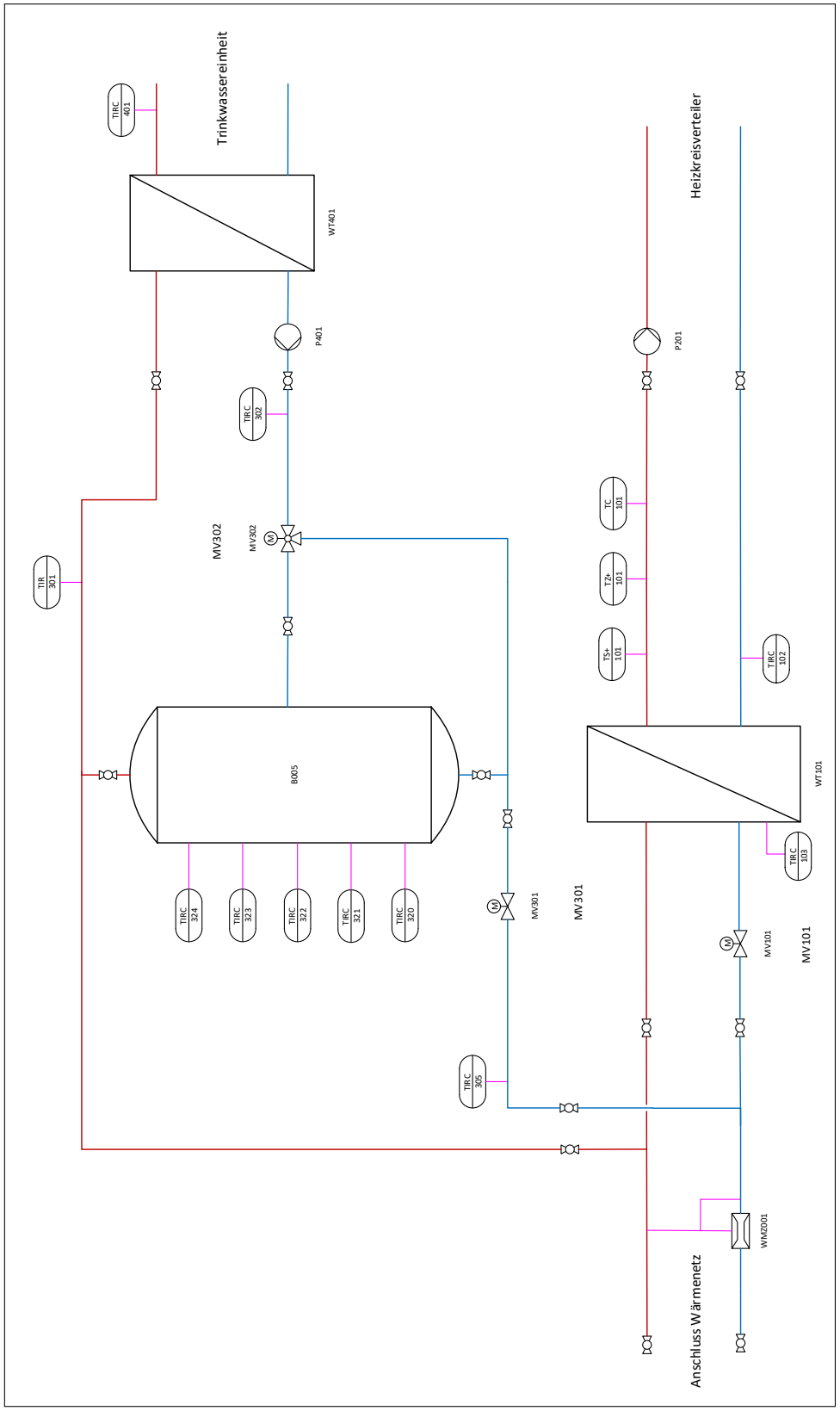


Abbildung 2.8.: R&I der iWÜST [11]

Tabelle 2.1.: Szenarien der iWÜST mit TWW

Szenario	**Beschreibung**
Inbetriebnahme	Prüfung aller Modulfunktionen, danach erfolgt Betriebsfreigabe
Minimale Temperatur	Sicherstellen einer minimalen TWW-Temperatur
Kommunikationsausfall	Keine Kommunikation mit der Leitwarte möglich; erhalten eines Basisbetriebs durch Vorgabe eines minimalen Speicherfüllstands
Netzkollaps	Das Netz ist instabil, d.h. die Sollvorlauftemperatur wird nicht erreicht. Netzstabilisierung durch minimale Abnahme
Funktionsprüfung	Sobald Mindestfüllstand überschritten ist, kann von der Energiezentrale jederzeit eine Funktionsprüfung für die Motorventile gestartet werden
Speicher überladen	Der Wärmespeicher wird maximal gefüllt, der Netzrücklauf wird möglichst aufgeheizt
Füllstandsbegrenzung	Begrenzung des Speicherfüllstandes, sobald die Temperaturdifferenz zwischen Speichervor- und -rücklauf zu klein wird
Leistungsbegrenzung	Übersteigt die in den letzten 15 Minuten bezogene Leistung die vertragliche Vereinbarung mit Kulanz, so wird die Leistungsaufnahme reduziert. Zunächst wird der Speicherfüllstand reduziert
RL-Begrenzung	Die Rückführung der Zirkulationstemperatur in den Rücklauf des Fernwärmenetzes wird mit einer Rücklaufbegrenzung verhindert
Normalbetrieb	Normalbetrieb; die Leitwarte sendet einen Füllstandsollwert, der kontinuierlich geregelt wird

Tabelle 2.2.: Szenarien der iWÜST ohne TWW

Szenario	**Beschreibung**
Inbetriebnahme	Prüfung aller Modulfunktionen, danach erfolgt Betriebsfreigabe
Warmhaltung	Minimale Ventilöffnung verhindert Auskühlen der Leitungsabschnitte
Kommunikationsausfall	Keine Kommunikation mit der Leitwarte möglich; erhalten eines Basisbetriebs durch Vorgabe eines minimalen Speicherfüllstands
Netzkollaps	Das Netz ist instabil, d.h. die Sollvorlauftemperatur wird nicht erreicht. Netzstabilisierung durch minimale Abnahme
Funktionsprüfung	Energiezentrale kann jederzeit eine Funktionsprüfung für die Motorventile starten
Rücklauftemperaturregelung	WÜST wird entsprechend einer definierten Rücklauftemperatur geregelt
Leistungsbegrenzung	Übersteigt die in den letzten 20 Minuten bezogene Leistung die vertragliche Vereinbarung mit Kulanz, so wird die Leistungsaufnahme reduziert.
RL-Begrenzung	Die Rückführung einer hohen Rücklauftemperatur ins Fernwärmenetz wird mit einer Rücklaufbegrenzung verhindert
Normalbetrieb	Normalbetrieb; der Temperatursollwert wird von der Energiezentrale vorgegeben

2.3. Speicherprogrammierbare Steuerung

SPS werden in vielen Bereichen für die Steuerung, Regelung und Überwachung von Anlagen und Systemen verwendet. Die zum Zwecke dieser Masterthesis verwendete SPS ist aus der Modellreihe 750 der Firma WAGO. Die folgenden Informationen beziehen sich somit auf das Modell 750-880, wobei ein Großteil dieser Eigenschaften auch für andere WAGO-Modelle und Modelle anderer Hersteller zutrifft.

2.3.1. Aufbau und Funktion

Die SPS besteht hauptsächlich aus einer zentralen Recheneinheit. Sie umfasst ein Mikroprozessorsystem, das betriebssystemgeführt den Programm- und Datenspeicher, Dateiein- und -ausgänge sowie den Programmablauf koordiniert [16]. Dies ist vereinfacht in Abbildung 2.9 dargestellt. Dank der modularen Bauweise der SPS können diese in vielen verschiedenen Bereichen zur Steuerung und Regelung von Systemen verwendet werden. Die SPS ist mit zwei Ethernetschnittstellen und einer IP-Adresse ausgestattet, worüber sie an ein Netzwerk angeschlossen und dann über eine Benutzeroberfläche konfiguriert werden kann.[17] Dabei werden Systemparameter (z.B. angeschlossene Klemmen, Belegung der Klemmen, eventuelle Zykluszeiten) bestimmt. Bei der in dieser Masterthesis verwendeten Benutzeroberfläche handelt es sich um die Entwicklungsumgebung *CoDeSys 2.3* des Softwareherstellers *3S-Smart Software Solutions GmbH*.

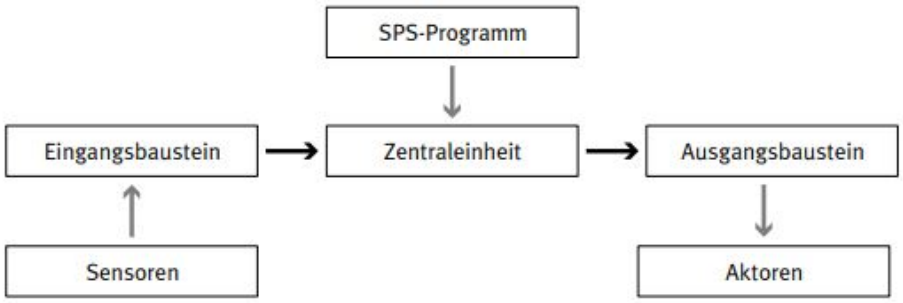


Abbildung 2.9.: Systemkomponenten einer SPS [16]

2.3.2. I/O-Module

Die Steuersignale von Aktoren und die Messsignale verschiedener Sensoren unterscheiden sich häufig hinsichtlich der Signalart (Strom, Spannung, Widerstandsmessung). Die Hersteller

Tabelle 2.3.: Beschreibung der verwendeten I/O-Module

I/O-Modul	Wertebereich (Anwenderprogramm)	Datentyp	I/O-Signal des Klemmanschlusses
Analogeingang für PT1000-Sensoren	−2000..8500	INT	0..3904 Ω
2-Kanal-Analogausgang	0..32 768	WORD	0..20 mA
4-Kanal-Analogausgang	0..32 768	WORD	DC 0..10 V
2-Kanal-Digitaleingang	TRUE/FALSE	BOOL	DC 0 V/24 V

bieten deshalb eine große Produktpalette an I/O-Modulen an, sodass sich der /die Nutzer/in die SPS mit den für die Anwendung benötigten Ein- und Ausgangsmodulen zusammenstellen kann. Module zum Einlesen und Ausgeben vom Sensor- und Aktorsignalen können dank eines Klemmsystems an die Recheneinheit in fast beliebiger Reihenfolge und Anzahl angebracht werden. Die Klemmen unterscheiden sich dabei hinsichtlich des zulässigen Stroms, der Spannung sowie der Umrechnung der Ein- und Ausgangssignale. Jede Klemme besitzt eine bestimmte Anzahl an Kanälen, welche die Anzahl der anschließbaren Sensoren bzw. Aktoren bestimmt. Die Klemmen sind über einen Feldbus mit der Recheneinheit verbunden.
Das Ein- bzw. Ausgangssignal wird in der Klemme in einen bestimmten Datentyp umgerechnet, welcher von der Recheneinheit der SPS verarbeitet bzw. ausgegeben werden kann. Klemmen können auch als serielle Schnittstellen fungieren, sodass Messgeräte mit seriellem Anschluss wie bspw. Wärmemengenzähler ausgelesen werden können. Dadurch entsteht eine hohe Variabilität bzgl. der Anschlüsse. Jede SPS kann somit für die benötigten Anforderungen belegt werden [18]. Die im Umfang dieser Thesis verwendeten I/O-Module sind in Tabelle 2.3 mit den relevanten Eigenschaften abgebildet.

2.3.3. Anwenderprogramm

Das Anwenderprogramm, welches auf die Recheneinheit der SPS gespielt wird, bestimmt im laufenden Betrieb, wie die angeschlossenen Aktoren auf die an den Eingängen anliegenden Signale reagieren sollen. Dabei kann das Programm in einer der folgenden fünf, durch die internationale Norm IEC61131-3 festgelegten Programmiersprachen erstellt werden.

- Anweisungsliste
- Kontaktplan
- Funktionsbaustein-Sprache
- Ablaufsprache
- Strukturierter Text

In diesen Sprachen sind Funktionen und Funktionsbausteine verfügbar, welche für das Erstellen des Programms verwendet werden können [19].

2.3.4. Export-Datei

In der Entwicklungsumgebung *CoDeSys 2.3* können bestehende Projekte inklusive aller enthaltenen Bausteine, Anwenderprogramme, Datentypen, Visualisierungen und Konfigurationen in einer Export-Datei des Formats *.exp* exportiert werden. Die Export-Datei kann mit einem Editor betrachtet werden. Sie ist in verschiedene Teile gegliedert, die durch bestimmte Codewörter voneinander untscheidbar sind. Beispielsweise werden Anwenderprogramme durch die Zeichenfolge *PROGRAM* und dem Programmnamen eingeleitet und durch die Zeichenfolge *END_PROGRAM* abgeschlossen. Innerhalb dieser Grenzen befinden sich die Deklarationen der Variablen und der Steuerungscode des Programms. Die Variablendeklaration ist ihrerseits widerum anhand der Zeichenfolgen *VAR* und *END_VAR* vom Steuerungscode unterscheidbar. Des Weiteren werden z.B. definierte Werte für Parameter von Bausteinen des Projekts bei dem Export in dem jeweiligen Codeteil in einer bestimmten Reihenfolge, aber ohne Codewörter, gespeichert. Anhand der Reihenfolge können die Zahlenwerten den Parametern zugeordnet werden. Im Anhang A.4 kann die Export-Datei des im Umfang dieser Arbeit erstellten Projekts eingesehen werden.

2.3.5. Strukturierter Text

Die im Umfang dieser Arbeit verwendete Programmiersprache der SPS ist ST. Sie ist eine textbasierte, an PASCAL angelehnte Programmiersprache, die hochsprachentypische Anweisungen (bspw. *IF...THEN...ELSE*) und Schleifen wie *WHILE..DO* ausführen kann [20]. Über Funktionsbausteine können z.B. Timer und PID-Regler in das Anwenderprogramm integriert werden. Das Programm besteht in der Regel aus einer Reihe von *Ausdrücken*, die sich aus Operanden und Operatoren zusammensetzen und nach der Auswertung einen Wert zurückgeben. Diese *Ausdrücke* werden nach vorgegebenen Bindungsregeln abgearbeitet [21].

2.3.6. Funktionsplan

Bei der Programmiersprache *Funktionsplan* (FUP) handelt es sich um eine grafisch orientierte Programmiersprache. Die Programme bestehen aus verschiedenen Logiksymbolen der boolschen Algebra. Die Logiksymbole repräsentieren bspw. Funktionsbausteine oder Operatoren, welche miteinander verbunden werden [21]. Diese Programmiersprache bietet

sich für Projekte mit einer begrenzten Anzahl von logischen Verknüpfungen an. Bei “größeren” Steuerungsaufgaben ist diese Sprache nicht vorteilhaft, da die grafische Darstellung dann zur Unübersichtlichkeit neigt

2.4. Parser

Das Wort “Parser” leitet sich von dem englischen Begriff “parse” ab, was zu deutsch “analysieren” und “zergliedern” bedeutet. Parser werden in der Informatik grundsätzlich verwendet, um Eingaben in ein anderes Format zu wandeln. Dies ist bspw. in Compilern notwendig, welche Code einer höheren Programmiersprache in Maschinencode wandeln.
Dabei wird ein sogenannter Lexer verwendet, um die als Eingabe definierte Zeichenfolge in verschiedene Token zu zerteilen. Token sind vordefinierte Zeichenketten, die im folgenden Kapitel 2.5 näher beschrieben werden. Durch eine ebenfalls vordefinierte Grammatik kann der Parser letztendlich an der Aneinanderreihung definierter Token bspw. Befehle erkennen und in das gewünschte Format übersetzen. Vereinfacht kann diese Methode für einen hypothetischen Parser zum Übersetzen vom deutschen in eine andere Sprache erläutert werden. Würde ein Parser für Sätze programmiert werden, so könnte der Satz “Der Computer rechnet.” wie folgt analysiert werden. Der Lexer scannt den Text und unterteilt ihn anhand der Leerzeichen in die Token “Der”, “Computer”, “rechnet’ und “.”’. Die Token, welche in diesem Fall Wörter und das Satztrennungszeichen sind, würden als Artikel, Nomen, Verb und Satzende erkannt werden. Grammatikalisch könnte der Parser dies dann als einen Hauptsatz erkennen und für eine weitere Verwendung, z.B. zum Übersetzen in eine andere Sprache, verarbeiten.

2.5. Pythonmodul PyParsing

Das Pythonmodul *PyParsing* wurde für die Programmierung des Parsers und des Importers verwendet. Dieses hatte sich gegenüber anderer Bibliotheken mit ähnlichen Funktionen aufgrund der angenehmen, klaren Handhabung für den Nutzer sowie der zugehörigen *MIT*-Lizenz durchgesetzt. Das Modul beruht darauf, dass durch eigens definierte Token Sprachen mit definierten Grammatiken erstellt werden können. Diese können genau auf die im zu untersuchenden Textdokument vorhandenen Zeichenketten zugeschnitten werden. In der Bibliothek sind dafür vorgefertigte Variablen, Funktionen und Klassen vorgesehen, die ein Arbeiten mit der Methodik erleichtern und die Lesbarkeit des Quelltextes verbessern [22].

Tabelle 2.4.: Verwendete Variablen der Bibliothek PyParsing

Variable	**Definierte Zeichen**
alphas	“ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz”
nums	“1234567890”
alphanums	“ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz 1234567890”
printables	“0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKL...”
empty	
lineStart	Anfang einer neuen Zeile
restOfLine	Rest der Zeile

Token

Token sind Zeichenketten, die mehr oder weniger genau definiert sind. Bei der Bestimmung der Token können Eigenschaften wie (maximale) Anzahl der Zeichen, enthaltene oder nicht enthaltene Zeichen oder das Vorhandensein von Großbuchstaben ausschlaggebend sein. Sie sollten so definiert sein, dass bei der Analyse des Textes nur die gesuchten Zeichenketten, Wörter oder Zahlen detektiert werden. Bei der Defintion der in dieser Masterarbeit verwendeten Token wurden die in Tabelle 2.4 gelisteten Variablen verwendet. Jeder Variable ist eine Vielzahl von Zeichen zugenordnet. Beispielsweise beinhaltet die Variable *alphas* alle Groß- und Kleinbuchstaben des lateinischen Alphabets. Diese könnte dann beispielsweise mit Hilfe einer Funktion mit einer Zeichenkette verglichen werden. Beinhalte die Zeichenketten ausschließlich Zeichen aus dem lateinischen Alphabet, so würde die Funktion eine Übereinstimmung der Variable mit der Zeichenkette zurückgeben. Im Code dieser Arbeit wurden 90 Token definiert. Jeder definierte Token ist mit dem Präfix “TK_” gekennzeichnet. Token können aus Variablen (z.B. *alphas*), definierten Zeichenketten (z.B. *“Temperatursensor”*) oder einer Kombination bestehen.

Klassen

Die Eigenschaften der Token können durch die von *PyParsing* zu Verfügung stehenden Klassen konkretisiert werden. So wird beispielsweise durch die Anwendung der Klasse *CaselessLiteral* auf das Token “var” definiert, dass die Groß- und Kleinschreibung unrelevant ist. Somit würden die Zeichenketten “VAR” und “vaR” eine Übereinstimmung mit dem Token ergeben. In Tabelle 2.5 sind die in dieser Arbeit genutzen Klassen darstellt.

Tabelle 2.5.: Verwendete Klassen der Bibliothek PyParsing

Klassenname	**Beschreibung**
ParserElement	Basisklasse der Parserfunktionen
Literal	Token, der mit einer bestimmten Zeichenkette genau übereinstimmt
Keyword	Token, der mit einer bestimmten Zeichenkette genau übereinstimmt. Es muss eines der Keyword-Zeichen (Leerzeichen oder "(") folgen.
CaselessLiteral	Wie Literal, aber Groß- und Kleinschreibung wird nicht beachtet
Word	Token, der ausschließlich aus den in Word definierten Zeichen besteht
White	Spezielle Klasse, die bei einem Leerzeichen übereinstimmt
LineStart	Übereinstimmend, falls die aktuelle Position am Anfang einer Zeile ist
And	Überprüft die Reihenfolge der aufeinander folgenden Token
Or	Einer der mit or verknüften Token muss mit der untersuchten Zeichenkette übereinstimmen
OneOrMore	Mindestens ein mal muss die folgende Sequenz oder der Token vorhanden sein
ZeroOrMore	Die folgende Sequenz oder der Token kann in belieber Anzahl oder gar nicht vorhanden sein
SkipTo	Zeichen werden übersprungen, bis die gesuchte Sequenz oder das Token gefunden wird.
Combine	Konverter, der die übereinstimmenden Token zu einem String verkettet
Group	Konverter, der die passenden Token als Liste zurückgibt
Suppress	Konverter, der die Ergebnisse des bearbeiteten Ausdrucks ignoriert
Optional	Als Optional definierte Token oder Sequenzen können, müssen aber nicht vorhanden sein

In der Klasse *ParserElement* sind eine Vielzahl von Methoden enthalten, von denen einige elementar für die Funktionalität des Parsers und des Importers sind. Im Code werden sie zum Suchen, Ersetzen und Ignorieren von übereinstimmenden Token und Sequenzen verwendet. Tabelle 2.6 zeigt die verwendeten Methoden dieser Klasse.

Funktionen

Neben den Variablen, Klassen und darin enthaltenen Methoden beinhaltet das Modul *PyParsing* Funktionen, die beim Erstellen eines Parsers verwendet werden können. Die in dieser Arbeit verwendeten Funktionen sind in Tabelle 2.7 gelistet.

Tabelle 2.6.: Verwendete Methoden der Klasse *ParserElement*

Klassenname	**Beschreibung**
setParseAction	Bei Übereinstimmung von untersuchter Zeichenkette und Token werden definierte Aktionen ausgeführt
parseString	Methode wandelt die übereinstimmende Zeichenkette in den gewünschten Ausdruck
scanString	Dokument wird bzgl. des gesuchten Token oder der Sequenz durchsucht. Rückgabewerte sind Anfang- und Endwert sowie der gesuchte Ausdruck
searchString	Wie scanString, aber mit veränderten Zugriffsmöglichkeiten auf die Rückgabewerte
transformString	Wie scanString, wobei die gefundene Zeichenkette durch eine modifizierte ersetzt wird
split	Methode trennt Zeichenketten bei definierten Zeichen
ignore	Definierte Ausdrücke werden ignoriert

Tabelle 2.7.: Verwendete Funktionen der Bibliothek PyParsing

Funktion	**Beschreibung**
oneOf	Hilfsfunktion zur Definition von mehreren zutreffenden Token
replaceWith	Token werden durch Aufrufen dieser Funktion durch definierte Zeichenketten ersetzt
nestedExpr	Hilfsfunktion, um verschachtelte Token zu finden

Grammatik

In der Grammatik sind die syntaktischen Beziehungen aufeinander folgender Token definiert. Es kann durch Konkatenation verschiedener Token eine Syntax definiert werden, nach der mit Hilfe der Methoden in Tabelle 2.6 gesucht werden kann. Diese Syntax beruht in dieser Arbeit auf der Programmiersprache ST. Die durch Konkatenation entstandenen Grammatiken werden in diesem Zusammenhang als Sequenz bezeichnet und im Code mit dem Präfix “SQ_” versehen.

2.6. Simulationsumgebung Jarvis

Die Simulationsumgebung Jarvis wird aktuell von dem Forschungsteam der HAW Hamburg im Projekt SHGH entwickelt. Sie ist ein auf der Programmiersprache Python basierendes, in viele Skripte unterteiltes Programm, welches Wärmenetze hinsichtlich der Drücke, Temperaturen und Massenströme für einen bestimmten Zeitraum simulieren können soll. Das Programm

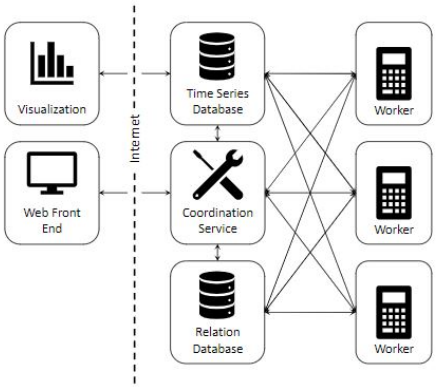


Abbildung 2.10.: Back-End-Architektur der Simulationsumgebung Jarvis [23]

wird als verteilte Back-End-Architektur strukturiert. Dies ist in Abbildung 2.10 dargestellt. Ein zentralisierter Koordinationsdienst ruft die für die Simulation notwendigen Skripte auf und sorgt für einen korrekten Simulationsablauf. In der relationalen Datenbank sind die benötigten Funktionen für die Berechnungen gespeichert. Eine Zeitreihendatenbank speichert die Ergebnisse aus den einzelnen Simulationsrechenschritten. Die einzelnen Dienste und Datenbanken können bei größeren Wärmenetzen auf unterschiedlichen Prozessoren und Rechensystemen (“Worker”) laufen, was die Berechnung umfangreicher und detaillierter Modelle ermöglicht. Die für die Berechnung der Simulationsparameter benötigten Prozesse laufen ebenfalls auf den Workern. Die aktuell verwendete grafische Benutzeroberfläche (GUI) wird demnächst von einer Weboberfläche als Front-End abgelöst. Die Simulationsergebnisse können mittels der webbasierten Applikation Grafana visualisiert werden.

2.6.1. Modellbildung in der Simulationsumgebung

In der Simulationsumgebung kann anwenderbezogen entschieden werden, ob die Systemdrücke in dem Modell des Wärmenetzes berücksichtigt werden. Bei Verzicht werden die Berechnungen nur anhand der Massenströme durchgeführt. Diese werden dabei von den Pumpen in den jeweiligen simulierten hydraulischen Leitungen vorgegeben. Misch-, Verteil- oder Durchgangsventile können dann nicht verwendet werden. Der Massenstrom in den jeweiligen Komponenten wird durch die Zuhilfenahme der Graphentheorie für das vorliegende System berechnet. Die Massenströme werden dabei, je nach Anzahl der geschlossenen Wasserkreisläufe, in einem oder mehreren sogenannten *Monolithen* berechnet. Enthält das Modell T-Stücke, so verwendet der Monolith diese als Knoten für die Graphentheorie. Die Verbindungen zwischen den T-Stücken sind die Kanten. Mit diesen Informationen wird dann die sogenannte *Kanten-Knoten-Matrix* gebildet, welche der Monolith zur Berechnung der Massenströme benötigt.[23]

Soll ein druckbehaftetes System modelliert werden, so wird für die Berechnung der Systemzustände der *Druckmonolith* verwendet. Dieser errechnet mit Hilfe eines iterativen Verfahrens die Drücke und die daraus resultierenden Massenströme.
Der/Die Anwender/in trifft die Entscheidung, ob ein druckbehaftetes oder ein druckloses Modell erstellt werden soll, indem er/sie die für den jeweiligen Fall vorgesehenen Komponenten bei der Modellbildung benutzt. Die Komponentenbibliothek enthält eine Vielzahl von Modellen, die hydraulische Bauteile aus der Wärmenetzinfrastruktur physikalisch und steuerungstechnische Bauteile logisch nachbilden. Durch die Benennung der Komponenten ist ersichtlich, welches Bauteil aus der Realität von der Komponente dargestellt wird. Dabei gibt es zu fast jedem Bauteil eine druckbehaftete und eine drucklose Komponente. Es ist auch möglich, dass die Simulationsergebnisse eines Teils der im System vorhandenen Wasserkreisläufe druckbehaftet und eines anderen Teils drucklos bestimmt werden.
Mit Hilfe der Komponenten kann das System in der gewünschten Ausführung entweder in der webbasierten grafischen Oberfläche (*Frontend*) oder in einem *yaml*-Skript erstellt werden. Dabei werden zunächst die im Modell enthaltenen Komponenten ausgewählt. Die Komponenten sind Objekte von Elementen aus der Komponentenbibliothek. Dadurch können mehrere Komponenten von einem bestimmten Typ verwendet werden.

Jede Komponente enthält eine bestimmte Anzahl an Ein- und Ausgangskonnektoren, die mit den jeweiligen Ein- und Ausgangskonnektoren einer anderen Komponente verbunden werden. Diese Verbindungen bilden die hydraulischen und elektronischen Anschlüsse nach. In jedem Zeitschritt der Simulation werden die physikalischen Systemzustände der einzelnen Komponenten eines Strangs, ausgehend von den T-Stücken, nacheinander berechnet (siehe Kapitel 2.6.2). Die in einer Komponente berechneten Werte werden dabei jeweils über die Konnektoren an die in Fließrichtung folgende Komponente weitergegeben, welche nach dem Empfangen der Werte mit der Berechnung startet. Dieses Vorgehen wird so lange durchgeführt, bis alle Systemzustände des Modells berechnet sind. Anschließend wird nach dem gleichen Verfahren für den nächsten Zeitschritt vorgegangen.
In den folgenden Beschreibungen der verwendeten hydraulischen Komponenten sind die Nachbarkomponenten bezüglich der Fließrichtung des Wärmeträgermediums definiert. Die Komponente, von der das beschriebene Objekt die Systemzustände erhält, wird als *Komponente_A* bezeichnet. Die vom Objekt berechneten Werte werden an *Komponente_B* gesendet.

2.6.2. Komponenten

Jede hydraulische Komponente enthält mindestens die Methoden *mass_flow* und *temperature*. In jedem Rechenschritt werden in diesen Methoden neue Zustandswerte der Systemkomponente bestimmt. Die Rechnung beruht bzgl. der Hydraulik und der Thermodynamik auf

physikalischen Modellen. Einige Komponenten enthalten Parameter, die für die Berechnung der Zustandswerte benötigt werden. Diese können bei der Modellbildung angepasst werden. Steuerungstechnische Komponenten wurden so programmiert, dass sie lediglich Werte für Signale über die Konnektoren empfangen und diese ohne weitere Berechnung an die angeschlossenen Komponenten schicken. Die steuerungstechnischen Komponenten wurden so entwickelt, dass sie die in der Realität verwendeten Bauteile hinsichtlich der Anzahl der Kanäle sowie der Signalart angemessen genau abbilden. Die im Umfang dieser Masterarbeit verwendeten Komponenten werden in der nachfolgenden Aufzählung näher beschrieben.

plc_cpu Das Element *plc_cpu* stellt die Steuereinheit der SPS dar. Sie empfängt als Eingangsvariablen die Abtastrate (*d_time*) sowie die Eingangssignale der angeschlossenen Klemmen über eine simulierte Busverbindung. Die Komponente ruft zu Beginn der Simulation das Skript *python_init_code* auf, in dem verschiedene Steuerungskomponenten wie Timer und PID-Regler initalisiert werden. In jedem Simulationsschritt wird das Skript *python_process_code* aufgerufen. Dabei werden die Signale der Klemmen eingelesen, das Anwenderprogramm wird vom Compiler abgearbeitet und die Ausgangssignale werden an die Klemmen geschickt. Diese Signale werden mit der fiktiven Busverbindung über die Klemmen an die zu regelnden Aktoren gesendet.

PLC_PT1000_Card Das simulierte I/O-Modul *PLC_PT1000_Card* empfängt Temperatursignale von angeschlossenen Komponenten in der Einheit Kelvin und leitet diese an die verbundene Komponente vom Typ *plc_cpu* weiter. Jedes Modul kann die Signale von maximal vier Temperatursensoren verarbeiten.

PLC_Analog_Output_Card Das Element *PLC_Analog_Output_Card* leitet Steuersignale, welche von einer Komponente des Typs *plc_cpu* berechnet wurden, an die angeschlossenen Aktoren des Systems weiter. Dabei ist darauf zu achten, dass das von der Steuereinheit generierte Signal im Wertebereich des angeschlossenen Aktors liegt.

PLC_End_Card Das Element entspricht der Endklemme einer SPS. Sie wird in der Realität benötigt, damit keine Kontakte des Busanschlusses der Klemmen offenliegen. Sie hat keine steuerungstechnische Funktion. In der Simulationsumgebung hat diese simulierte Klemme ebenfalls keine steuerungstechnische Funktion, jedoch ist ein Starten der Simulation ohne Verwendung dieser Klemme nicht möglich.

Temperature_Sensor Die Temperatursensoren sind als Komponenten in die hydraulischen Wasserkreisläufe eingebaut. Die Methode *temperature* schickt die Temperatur in der Einheit Kelvin über die als Ausgang definierten Konnektoren an die angeschlossenen Komponenten. Der Massenstrom aus Komponente_A wird an die Komponente_B gesendet.

Pump_control_signal Komponenten vom Typ *Pump_control_signal* berechnen in der Methode *mass_flow* anhand des maximalen und minimalen Massenstroms und des Steu-

ersignals den Massenstrom der Kante. Das Steuersignal erhält die Pumpe von einer Komponente des Typs *PLC_Analog_Output_Card*. Der Wert der Temperatur aus der Komponente_A wird ohne Veränderung an die Komponente_B weitergegeben.

Pump Der durch die Parametrierung der Komponente festgelegte Massenstrom wird in der Methode *mass_flow* bestimmt und mit der unveränderten Temperatur der Komponente_A an die Komponente_B weitergereicht.

Load_From_Timeseries Die Leistungsabnahme der nachgebildeten Last wird von einer Zeitreihe bestimmt, der ein Leistungswert in der Einheit Kilowatt hinterlegt ist. Während der Simulation berechnet die Methode *temperature* aus der Vorlauftemperatur und dem Massenstrom der Komponente_A sowie der im aktuellen Zeitschritt hinterlegten Leistung und dem c_p-Wert anhand Gleichung 2.3 die Rücklauftemperatur der Last, die an die Komponente_B mit dem unveränderten Massenstrom weitergegeben wird.

Heat_Exchanger Die Methode *temperature* des Elements *Heat_Exchanger* berechnet anhand der übergebenen Werte der Eingangstemperaturen und Massenströme sowie der definierten Parameter die Ausgangstemperaturen des reinen Gegenstromwärmeübertragers, indem das Modell des Wärmeübertragers in einzelne Zellen unterteilt und für jede Zelle die übertragene Wärme berechnet wird [23]. Die Ausgangstemperaturen und die unveränderten Massenströme werden an die Komponente_B_prim des Primärkreislaufs und an die Komponente_B_sek des Sekundärkreislaufs übergeben.

CHPPlant Objekte vom Typ *CHPPlant* berechnen bei Aufruf der Methode *temperature* die Ausgangstemperatur nach Formel 2.3. Die Wärmemenge ist abhängig vom Steuersignal und der maximalen Leistung. Die Eingangstemperatur und der Massenstrom werden von der Komponente_A gesendet. Der berechnete Wert der Temperatur sowie der unveränderte Massenstrom wird an die Komponente_B geschickt.

Storage_2ConnectionPieces Schichtenspeicher vom Typ *Storage_2ConnectionPieces* berechnen die Temperaturverteilung mit Hilfe eines *Kanten-Knoten-Modells* [23]. Durch die Parametrierung können Wärmeverluste und Anzahl der Temperaturschichtungen definiert werden. Je nach Anzahl der vorgegebenen Schichten können genauso viele Temperaturen im Speicher ausgelesen werden. Objekte vom Typ *Storage_2ConnectionPieces* enthalten zwei Konnektoren, über die sie Be- und Entladen werden.

T-Piece Komponenten vom Typ *T-Piece* besitzen drei hydraulische Anschlüsse und können in alle Richtungen durchflossen werden. Die resultierenden Massenströme in der Komponente werden vom Monolith berechnet (siehe Kapitel 2.6.1). In der Methode *temperature* werden die daraus resultierenden Temperaturen berechnet.

Pipe Der Temperaturverlust in der simulierten Rohrleitung wird durch die Methode *temperature* berechnet. Anschließend wird diese und der unveränderte Massenstrom an die Komponente_B weitergegeben.

2.6.3. Regelung in der Simulationsumgebung

In der Simulationsumgebung sind mehrere Regler in Form von Klassen enthalten. Die Regelung von Aktoren kann durch Objekte der jeweiligen Klassen erfolgen und wird in den folgenden Kapiteln beschrieben.

Klasse PID

Bei der Initialisierung der Objekte der Klasse *PID* im Skript *python_init_code* können Parameter definiert werden, die denen der PID-Regler aus der Entwicklungsumgebung *CoDeSys 2.3* entsprechen. Der Regler wird im *python_process_code* aufgerufen. Im hinterlegten Code der Funktion *calculate_control_value* wird anhand der Parameter und dem aktuellen Ist- und Sollwert die Stellgröße berechnet und zurückgegeben. Dieser Wert kann anschließend an den Aktor über eine simulierte Klemme gesendet werden. Folgende Werte werden an den Regler gesendet bzw. können bei der Reglereinstellung parametriert werden: [21]

d_time Taktung, in der der Regler aufgerufen wird. Wird bei der Initalisierung des Simulationsmodell vorgegeben.

ACTUAL Istwert der Regelgröße (Temperaturen der Sensoren bzw. im Anwenderprogramm berechneter SOC).

SET_POINT Sollwert; Wird von Leitwarte als Solltemperatur oder Soll-SOC vorgegeben. Bei Eintreten von bestimmten Szenarien bestimmen diese den Sollwert.

KP Verstärkungsfaktor des P-Anteils

TN Nachstellzeit zur Berechnung des I-Anteils

TV Vorhaltezeit zur Berechnung des D-Anteils

Y_MANUAL Bei Setzen von *MANUAL* = *TRUE* wird der in Y_MANUAL hinterlegte Wert als Ausgangssignal ausgegeben.

Y_OFFSET Offset des Ausgangssignal

Y_MIN, Y_MAX Untere bzw. obere Grenze des Ausgangssignals. Bei Überschreiten der Grenze wird Stellgröße in der Grenze gehalten.

MANUAL Aktiviert die manuelle Stellgrößenvorgabe

RESET Durch Setzen von *RESET* auf *TRUE* werden alle Werte reinitalisiert.

Klasse AnalogDriveController

Objekte der Klasse *AnalogDriveController* sorgen dafür, dass sich die Steuersignale, welche an die Aktoren des Modells gesendet werden, in den parametrierten Grenzen befinden. Den Objekten werden maximale und minimale Ausgangssignale zugewiesen. Der in der Funktion *calculate_position* hinterlegte Code greift bei Über- oder Unterschreiten dieser Grenzwerte ein und gibt die Grenzwerte aus.

3. Parser und CODESYS-Importer

In diesem Kapitel wird der im Zusammenhang mit dieser Masterthesis erstellte Parser beschrieben. Zunächst wird darauf eingegangen, weshalb in der Simulationsumgebung *Jarvis* ein Parser benötigt wird. Anschließend werden verschiedene Programme vorgestellt, welche untersucht wurden, ob mit Hilfe dieser die Anforderungen erfüllt werden könnten. Es wird dann der Grund beschrieben, weshalb entschieden wurde, dass der Parser händisch programmiert wurde. Der grundlegende Aufbau des Programmes, die programmiertechnische Umsetzung mit dem Pythonmodul *PyParsing* sowie die Ablaufroutine der enthaltenen Methoden werden erklärt und anhand eines Codeteils erläutert. Im Anschluss wird das Einlesen sowie die Verwendung der *CoDeSys 2.3* -Exportdatei und die Integration in *Jarvis* beschrieben. Abschließend werden die einzelnen Funktionen des Parsers sowie die Funktion zum Erstellen der Skripte dargelegt.

3.1. Zweck des Parsers

Die in Kapitel 2.6 beschriebene Simulationsumgebung *Jarvis* soll ein reales Wärmenetzsystem als physikalisches Modell abbilden. Dazu gehört neben den physikalischen Eigenschaften der Komponenten die Regelung. Es wurden dafür Modelle gebaut, die die Steuerung bei angemessenem Aufwand so exakt wie möglich darstellen. Es können in der Simulation neben der Recheneinheit (*CPU*) verschiedene Klemmen ausgewählt werden, welche mit denen der Baureihe 750 der Firma WAGO bezüglich der Anschlüsse übereinstimmen. Dadurch wird ermöglicht, dass letztendlich das Fernwärmesystem inklusive aller Hardwarekomponenten sowie der Regelungsabläufe beschrieben werden kann.

In Kapitel 2.3 wurden die unterschiedlichen Programmiersprachen einer SPS benannt. Diese können vom Compiler der pythonbasierten Simulationsumgebung *Jarvis* nicht gelesen werden. Üblicherweise wird in der industriellen Umsetzung von Regelungen der Code der verschiedenen Steuerungen von Personen geschrieben, welche auf den Umgang mit der Programmiersprache sowie der Entwicklungsumgebung für SPS spezialisiert sind. Sie könnten bei Verwendung der Simulationsplattform für die Auslegung eines Wärmenetzes mit Hilfe von *Jarvis* die Regelung in der für sie bekannten Umgebung (bspw. *CoDeSys 2.3*) erstellen. Die Steuerung könnte dann mit Hilfe einer Exportdatei in *Jarvis* geladen werden.

Der Parser übersetzt diesen Code dann in die Programmiersprache Python, sodass die Simulationsumgebung den Code verwenden kann. Somit muss bei der Verwendung des Anwenderprogramms bei gleichen Eingangsvariablen das exakt gleiche Ergebnis in ST und Python generiert werden. Für ein einfaches Übertragen der Simulationserkenntnisse und die dafür erstellten Regelungsabläufe in die realen Systeme wäre es also hilfreich, wenn der Code der Steuerung in einer Sprache geschrieben und automatisch in eine andere übersetzt werden könnte. Dadurch würden Fehler und Zeitaufwand minimiert werden. Für diese Zwecke wurde der im Folgenden beschriebene Parser erstellt.

3.2. Recherche zu existierenden Programmen

Zunächst wurde untersucht, ob frei zugängliche Programme existieren, mit denen die oben beschriebenen Anforderungen erfüllt werden können. Da bei der Recherche kein Programm gefunden wurde, das Code direkt von ST in Python oder invers wandeln kann, wurde in Erwägung gezogen, dies mit einem Zwischenschritt in der Sprache *C* oder *xml* durchzuführen. Bei der Suche zu Programmen mit *xml*-Dateien wurden nur solche berücksichtigt, die den Standard *PLCOpen*[1] verwenden.

3.2.1. Programmauswahl zum Wandeln von Python in ST

Folgende Programme wurden bzgl. ihrer Möglichkeiten zum Übersetzen von Python in ST untersucht. Die generelle Vorgehensweise (siehe Abbildung 3.1) wäre dabei, dass der Steuerungsalgorithmus in Python geschrieben und bspw. mittels der Bibliothek *ElementTree* in eine *xml*-Datei gewandelt werden würde. Für die weitere Verwendung müsste dieser Code dann mit einem Programm in die Form *PLCOpen* gebracht werden, welches selbst geschrieben werden müsste. Anschließend könnte die *PLCOpen-xml*-Datei mit einem der folgenden Programme in ST gewandelt werden.

Beremiz Das frei verfügbare Programm *Beremiz* ist in der Lage, xml-Dateien auf *PLCOpen*-Basis in ST zu wandeln. Die xml-Datei müsste dann jedoch im PLCOpen-Format vorliegen. Somit wäre ein zusätzlicher Programmierschritt von Nöten, um die *xml*-Datei in *PLCOpen-xml*-Datei zu ändern.

CODESYS V3 Die Entwicklungsumgebung *CODESYS V3* ermöglicht ein Einlesen von *PLCOpen*-Dateien und kann diese verwenden. Es wurde angenommen, dass der Code dann als ST vorläge und eine weitere Verwendung in *CoDeSys 2.3* ermöglichen

[1] PLCOpen: Organisation, die verschiedene Standards aus dem Umfeld der Steuerungtechnik in herstellerunabhängige Formate zusammenfasst

würde. Dabei müsste jedoch, wie im Fall oben, die *xml*-Datei aus dem Pythondokument erstellt werden, was einen hohen Arbeitsaufwand bedeutet. Außerdem müsste *CODESYS V3* kostenpflichtig erworben werden.

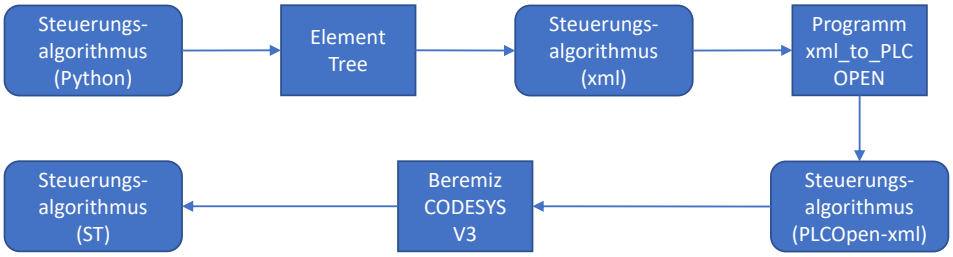


Abbildung 3.1.: Flussdiagramm zur Verwendung zum Übersetzen von Python in ST

3.2.2. Programmauswahl zum Wandeln von ST in Python

Bei der Recherche zur Wandlung von ST- in Python-Code wurden verschiedene Programme untersucht. Es konnte kein Programm gefunden werden, welches die Umwandlung direkt durchführt. Deshalb wurde eroiert, ob die Umwandlung mit Hilfe der folgenden Programme umsetzbar ist. Dabei soll zunächst von den Programmen *Beremiz* oder *GEB Automation* aus ST-Code C-Code erstellt werden, welcher dann in einem zweiten Schritt in Python-Code gewandelt werden würde (siehe Flussdiagramm in Abbildung 3.2). Bei der Validierung der Programme *Beremiz* und *GEB Automation* wurden diese kostenfrei heruntergeladen und zur Wandlung eines kurzen Codes in ST getestet. In der folgenden Aufzählung werden zunächst die beiden Programme beschrieben, die C-Code erstellen und anschließend Möglichkeiten zur Wandlung oder Integration von C-Code in Python vorgestellt.

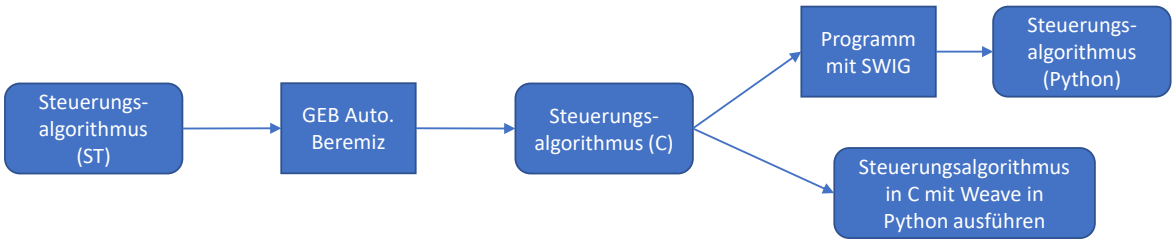


Abbildung 3.2.: Flussdiagramm zum Übersetzen von ST in Python

GEB Automation Das Programm *GEB Automation* wandelt Code der Norm IEC61131-3 in C-Code. Es besitzt eine übersichtliche Benutzeroberfläche und ist kostenfrei erhältlich. Der vom Programm generierte C-Code besteht aus mehreren, voneinander abhängigen und verschachtelten Funktionen. Der Code ist nicht ohne weitere Zwischenschritte für die gewünschte Nutzung verwendbar.

Beremiz Das in Kapitel 3.2.2 genannte Programm Beremiz kann Eingaben der IEC61131-3-Norm in C-Code wandeln. Der dabei generierte Code ist, ähnlich wie bei dem Erzeugnis von *GEB Automation*, im Bezug auf den eingelesenen ST-Code und die Weiterverwendung zu komplex.

Weave Das *Weave*-Paket der *Scipy*-Bibliothek ermöglicht das Aufrufen und Abspielen von einzelnen Funktionen und Codes in der Pythonumgebung. Dabei führt die *inline*-Funktion den Code aus. Das Paket kann nicht ohne weitere Anpassungen Code mit mehreren Funktionen und Abhängigkeiten ausführen.

Swig *Swig* ist ein *Open Source*-basiertes Programmierwerkzeug, welches als Schnittstellencompiler C-Programme mit Skriptsprachen wie Python verbindet. Das Werkzeug unterstützt jedoch nicht die Verwendung von *Arrays* im C-Code. Arrays sind jedoch ein Bestandteil der meisten Anwenderprogramme in SPS, weshalb sich gegen die Verwendung von *Swig* in diesem Kontext entschieden wurde.

Die Recherche zu den Möglichkeiten zur Umwandlung von ST in Python bzw. Python in ST zeigt, dass eine kostenlose und elegante Lösung bei Verwendung von vorhandenen Programmen oder Bibliotheken nicht möglich ist. Aufgrund dessen wurde entschieden, dass ein Parser für die Umwandlung von ST in Python eigenständig programmiert werden soll.

3.3. Programmiertechnische Umsetzung

In diesem Kapitel wird aufgezeigt, wie der Parser sowie der Importer für *CoDeSys 2.3*-Exportdateien in Python umgesetzt wurde.

3.3.1. Aufbau

Der Programmcode ist in das Skript *plc_library* der Simulationsumgebung *Jarvis* integriert. Er besteht aus den beiden Klassen *STParser* und *CODESYSImporter* sowie den Funktionen *create_init_main* und *get_python_code*. Anhand des Flussdiagramms in Abbildung 3.3 können die Aufgaben der Klassen und der Funktion *create_init_main* bei Einbindung von ST-Code erläutert werden. Beim Erstellen des Modells wird in der grafischen Benutzeroberfläche (GUI) der Simulationsumgebung das Feld *PLC Editor* ausgewählt. Die in *CoDeSys 2.3* erstellte

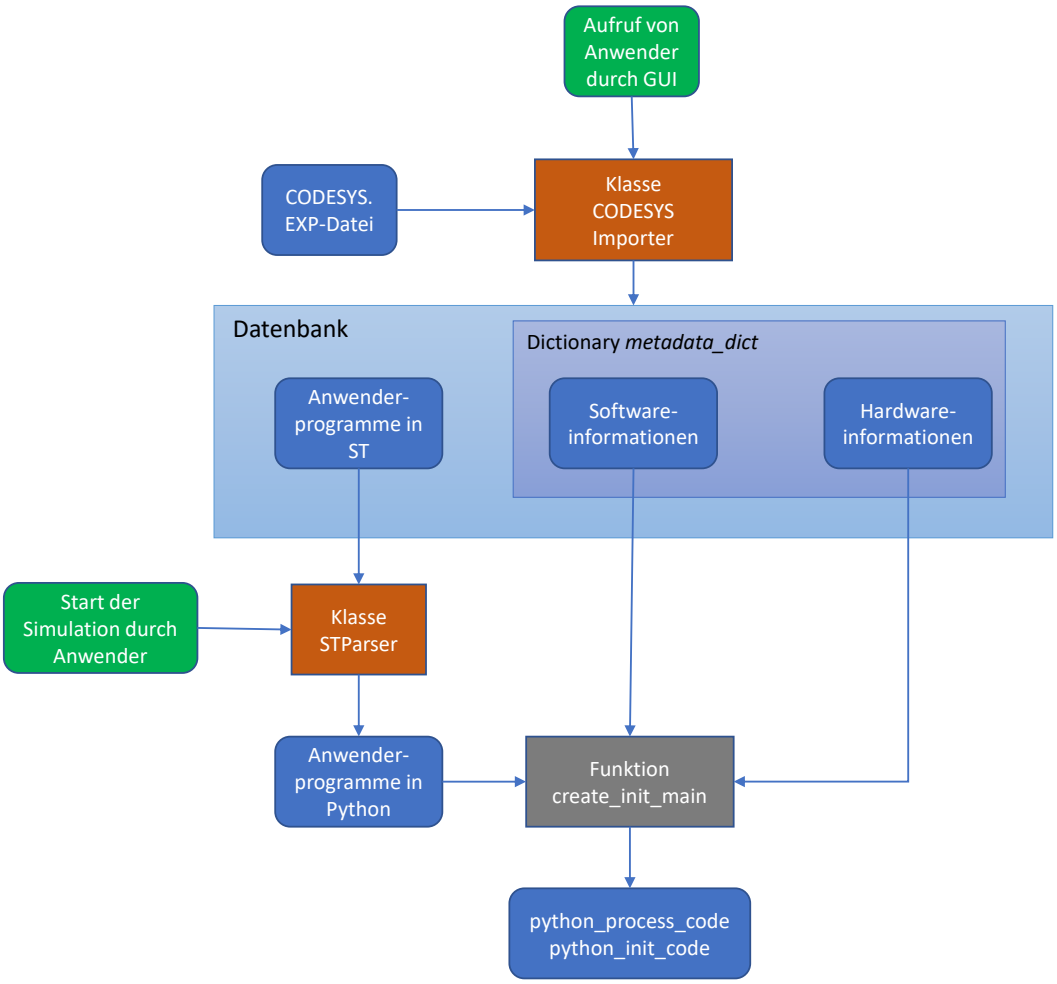


Abbildung 3.3.: Flussdiagramm von Importer und Parser

Export-Datei mit den Steuerungsprogrammen und -konfigurationen kann anschließend ausgewählt werden. Beim Öffnen der Datei wird diese durch die Klasse *CODESYSImporter* in die für die Weiterverarbeitung relevanten Komponenten aufgeteilt und als Dictionary in der Datenbank als *metadata_dict* gespeichert. Dieser Schritt wird in Kapitel 3.3.3 näher beschrieben. Dieses Dictionary beinhaltet Hard- und Softwareeinstellungen des Steuerungssystems. Die für den Regelungsablauf definierten Anwenderprogramme werden ebenfalls in der Datenbank gespeichert und können bei Bedarf nach dem Import angepasst werden.
Durch den Start der Simulation werden zunächst, falls der Code in ST vorliegt, die Anwenderprogramme in ST mit Hilfe der Klasse *STParser* in Python übersetzt. Die dafür verwendeten Methoden sowie deren programmiertechnische Umsetzung werden im Folgenden beschrieben. Danach erstellt die Funktion *create_init_main* aus den im Dictionary gespeicherten Daten die für den Simulationsablauf benötigten Skripte *python_process_code* und *python_init_main*,

Tabelle 3.1.: Zusammenfassung der programmiertechnischen Bestandteile des Importers und des Parsers

Bestandteil	**Funktion**
CODESYSImporter	Importiert ausgewählte Export-Datei, extrahiert und speichert relevante Informationen in verschiedene Dictionaries
STParser	Wandelt Anwenderprogramme aus der Export-Datei der Sprache ST in Python
metadata_dict	Enthält die extrahierten Daten aus der Export-Datei
create_init_main	Erstellt aus Daten (*metadata_dict*) und Anwenderprogrammen in Python die Skripte *python_process_code* und *python_init_code*
python_init_code	Skript, in dem die verwendeten Timer und Regler des Anwenderprogramms initialisiert werden.
python_process_code	Skript, welches den Regelalgorithmus der Steuerung enthält

welche im Kapitel 3.3.5 näher beschrieben werden. Diese werden ebenfalls in der Datenbank gespeichert. Die Funktion *get_python_code* lädt den Code des Anwenderprogramms in die Simulationsumgebung, wenn dieser in der Sprache Python verfasst wurde und als Skript vorliegt. In Tabelle 3.1 sind die beschriebenen Klassen, Funktionen, Dictionaries und Skripte zusammengefasst.

3.3.2. Verwendung des Pythonmoduls Pyparsing

Die Verwendung des Moduls *PyParsing* im Umfang dieser Masterarbeit wird in den folgenden zwei Abschnitten anhand der statischen Methode *get_programm* und *parse_limit* erklärt.

Detektion von Token und Sequenz

Der relevante Teil des Codes der Methode *get_programm* ist in Quellcode 3.1 einsehbar. Dieser ist durch das Ablaufdiagramm in Abbildung 3.4 visualisiert. Die beschriebene Methode soll die in der Export-Datei vorhandenen Anwenderprogrogramme extrahieren und in das im Dictionary *metadata_dict* unterlagerte Dictionary *programs_dict* mit der Bezeichnung des Programmes speichern. Der im Folgenden beschriebene Code sammelt die Start- und Endpunkte der gesuchten Programmpassagen in der Exportdatei in Listen (*array_program_end*, *array_program_start*). Außerdem wird der Name des Anwenderprogramms in einer weiteren Liste gespeichert (*array_program_name*). Das Suchen nach den definierten Token wird von der Methode *scanString* (siehe Tabelle 2.6) durchgeführt.

Quellcode 3.1: Quellcode der Methode *get_program*

```
@staticmethod
    def get_program(codesys_export_file):
        """
        Extract all programs (main_loop(s), PIDs,...) from the CODESYS file
        :param codesys_export_file: CODESYS export file (*.exp)
        :return: dict of all programs with the related code
        """
        array_program_end = []
        array_program_start = []
        array_program_name = []
        for match, start, end in CODESYSImporter.TK_PROGRAM.scanString(codesys_export_file):
            array_program_start.append(end)
            array_program_name.append(match)

        for match, start, end in CODESYSImporter.TK_PROGRAM_END.scanString(codesys_export_file):
            array_program_end.append(start)

        i = len(array_program_end)
        k = 0
        list_programs = []
        list_names = []
        while k < i:
            list_programs.append((codesys_export_file[array_program_start[k]:array_program_end[k]]))
            list_names.append(array_program_name[k][0][1])
            k += 1
        programs_dict = dict(zip(list_names, list_programs))
        return programs_dict
```

Zunächst werden die benötigten Listen initialisiert. Die darauf folgende for-Schleife durchsucht die Export-Datei *codesys_export_file* nach dem Token *TK_PROGRAM* aus der Klasse *CODESYSImporter*. Das Token *TK_PROGRAM* wird zu Beginn der Klasse folgendermaßen definiert:

TK_PROGRAM = pp.Group(pp.Literal("\nPROGRAM") +STParser.TK_VARIABLE)

Das Token *TK_VARIABLE* der Klasse *STParser* ist definiert als:

TK_VARIABLE = pp.Optional(pp.Word("(")) + pp.Combine(pp.Optional(") + pp.Word\ (pp.alphanums + "_"+ "."+ "["+ "]"+ “")) + pp.Optional(pp.Word(")"))

Das Token TK_VARIABLE wird verwendet, um im Code genutzte Variablen bzw. Benennungen zu detektieren. Somit sucht die *for*-Schleife nach Zeichenketten, bei denen auf das Token *“PROGRAM”* eine Bezeichnung für das Programm folgt. In der Export-Datei sind Anwenderprogramme immer durch diese Syntax gekennzeichnet. Bei jedem gefundenen Anwenderprogramm wird dann die Stelle des Endpunktes der Definition als Integer und der Name des Programms als Zeichenkette in einer Liste gespeichert. Ein Programm wird mit dem Token *TK_PROGRAM_END* abgeschlossen. Die zweite *for*-Schleife sucht und speichert den Ort des Endes des Anwenderprogramms mit Hilfe dieses Tokens in eine weitere Liste. Durch die in den Listen abgelegten Start- und Endpunkte ist der Bereich definiert, in dem ausschließlich das Anwenderprogramm vorliegt. Dieser Bereich wird dann im weiteren Programm mit Hilfe der Listen in das Dictionary gespeichert.

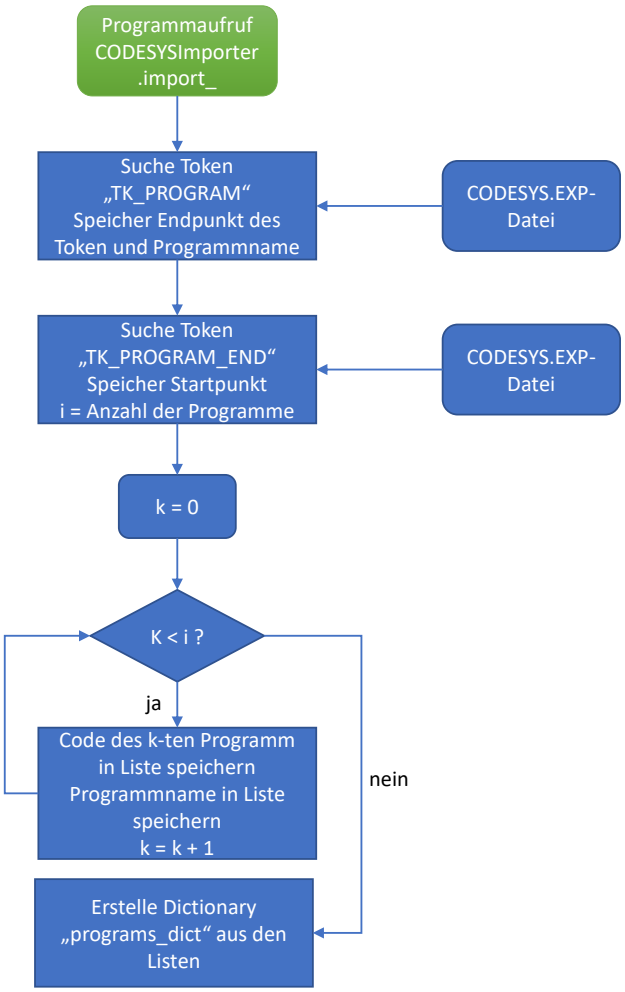


Abbildung 3.4.: Programmaublaufplan *get_program*

Umwandeln der ST-Syntax in Python-Syntax

In diesem Abschnitt wird die grundsätzliche Funktion der statischen Methoden der Klasse *STParser* anhand der Methode *parse_limit* beschrieben. Jede Methode ist für die Wandlung einer ST-spezifischen Syntax zu Python zuständig oder bereitet die Zeichenketten für die Umwandlung auf. Die Methoden werden nacheinander auf das Programm angewendet. Die Methode *parse_limit* ändert die in ST verfügbare Funktion *LIMIT* in eine min/max-Abfrage, welche von Pythoncompilern gelesen werden kann. Der Methodencode ist in Quellcode 3.2 gezeigt. Abbildung 3.5 zeigt das zugehörige Ablaufdiagramm.

Quellcode 3.2: Programmcode der Methode parse_limit

```
@staticmethod
    def parse_limit(hybrid_code):
        """
        Convert limit function from ST-Code to min/max request in Python
        :param hybrid_code: program code of the parsed code in structured text
        :return:hybrid_code: processed program code of the parsed code in structured text
        """
        array_end_limit = []
        for match, start, end in STParser.SQ_LIMIT.scanString(hybrid_code):
            array_end_limit.append(end)

        i = len(array_end_limit) - 1
        k = 0
        array_end_limit.insert(0, 0)

        text_recapped = ""
        while k <= i:
            text_split_sel = pp.SkipTo(STParser.SQ_LIMIT, include=True).parseString(hybrid_code\
            [(array_end_limit[k]):])
            ret_var = text_split_sel[1]
            input_var = text_split_sel[5]
            limit_min = text_split_sel[7]
            limit_max = text_split_sel[9]
            replace_sequence = (ret_var + "= min(max(" + input_var + "," + limit_min + ")," +\
            limit_max + ")")
            text_recapped = text_recapped + text_split_sel[0] + replace_sequence
            k += 1

        hybrid_code = text_recapped + (hybrid_code[array_end_limit[k]:])
        return hybrid_code
```

Zunächst wird, wie im vorherigen Abschnitt beschrieben, nach der Sequenz *SQ_LIMIT* im Anwenderprogramm (*hybrid_code*) gesucht und das Ende der Sequenz(en) in einer Liste (*array_end_limit*) gespeichert. Der erste Wert der Liste wird als 0 definiert, da jedes Argument der Liste einen Startpunkt der Suche zu Beginn der while-Schleife definiert und die erste Suche am Anfang des Anwenderprogramms starten soll. Die Zählvariablen *i* und *k* sowie die Zeichenkette *text_recapped* werden initialisiert. Die while-Schleife wird so oft durchlaufen, bis alle Sequenzen der Syntax *LIMIT* bearbeitet wurden. In der Schleife wird mittels *SkipTo* die Sequenz gesucht. Diese wird dann durch die Methode *parseString* als Liste *text_split_sel* gespeichert. Die Argumente von *LIMIT* werden als einzelne Listenelemente gespeichert. Anschließend werden diese in neuer Form als *replace_sequence* gespeichert, welche den gleichen Ausgabewert in Python wie *LIMIT* in ST generiert. Anschließend wird die ersetzende Zeichenkette im Programmcode eingefügt. Nachdem alle Sequenzen umgewandelt wurden wird der komplette Code als *hybrid_code* zusammengesetzt und zurückgegeben. Die meisten Methoden der Klasse *STParser* agieren nach diesem Suchen-und-Ersetzen-Schema.

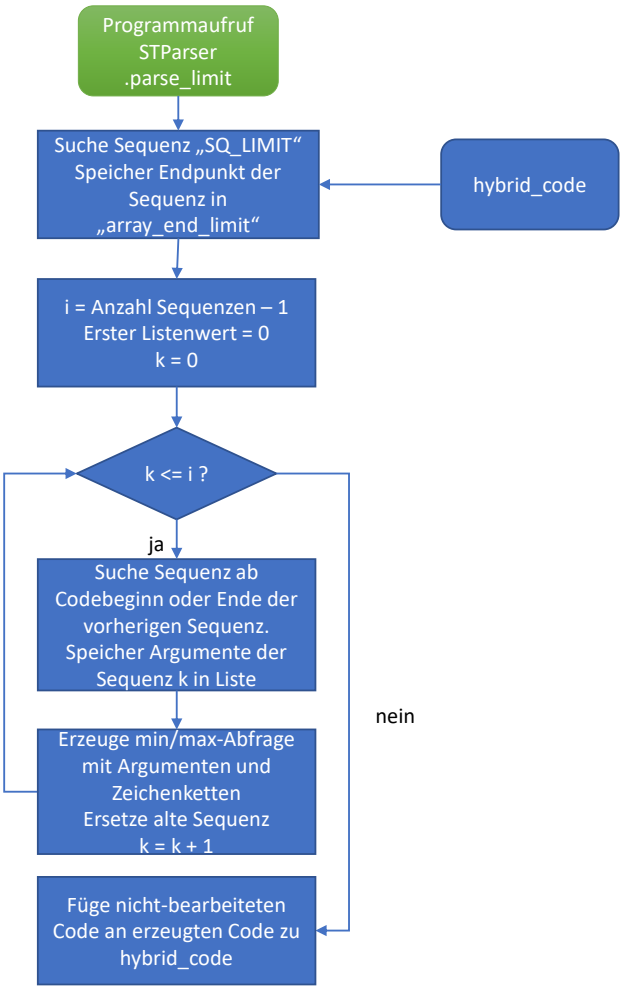


Abbildung 3.5.: Programmaublaufplan *parse_limit*

Umwandlung der Operatoren

Bei der Umwandlung von Operatoren in ST zu Python wurde in den meisten Fällen mit einer Kombination der Funktionen *transformString* und *replaceWith* gearbeitet. Dabei wird der Text nach den Operatoren von ST in Tabelle 3.2 durchsucht und, falls diese von dem Symbol in Python abweicht, durch die zugehörigen Operatoren aus Python im Code ersetzt. Bei der Wandlung von Funktionen wie *sin* oder *ln* wurden Methoden der Pythonbibliothek *math* verwendet.

Tabelle 3.2.: Vergleich der Operatoren und mathematischen Funktionen in ST und Python

Operator/Funktion	**Strukurierter Text**	**Python**
Negation	-	-
Einzel-Plus	+	+
Komplement	NOT	~
Potenzieren	**	**
Multiplizieren	*	*
Dividieren	/	/
Modulo	MOD	%
Addieren	+	+
Subtrahieren	-	-
Vergleich	<,>,<=,>=	<,>,<=,>=
Gleichheit	=	==
Ungleichheit	<>	!=
Boolesches UND	&, AND	&, and
Boolesches Exklusives ODER	XOR	^
Boolesches ODER	OR	or
Linksschieben	SHL	«
Rechtsschieben	SHR	»
Adressoperator	ADR	id
Zuweisung	:=	=
Wurzel	SQRT	math.sqrt
Natürlicher Logarithmus	LN	math.log
Zehnerlogarithmus	LOG	math.log10
Exponentialfunktion	EXP	math.exp
Sinus	SIN	math.sin
Cosinus	COS	math.cos
Tangens	TAN	math.tan
Arcussinus	ASIN	math.asin
Arcuscosinus	ACOS	math.acos
Arcustangens	ATAN	math.atan

3.3.3. Klasse CODESYSImporter

In diesem Abschnitt wird erläutert, wie bestimmte Daten mit Hilfe der statischen Methoden und Funktionen der Klasse *CODESYSImporter* aus der Export-Datei extrahiert werden. Die Informationen werden im Dictionary *metadata_dict* in eigenen Dictionaries gespeichert. Diese Informationen werden für die Erstellung der Skripte *python_process_code* und *python_init_code* genutzt. Einige Informationen werden aktuell noch nicht für die Weiterverarbeitung genutzt, jedoch wurden sie für eventuelle Anpassungen oder Erweiterungen trotzdem in das Dictionary aufgenommen. Durch Aufrufen der Funktion *codesys_import_to_database* bzw. *python_import_to_database* wird die Funktion *import_* aufgerufen. In dieser Funktion werden nacheinander statische Methoden aufgerufen, welche die Informationen aus der Export-Datei entnehmen und in die Dictionaries und Listen speichern. Anschließend werden die extrahierten Anwenderprogramme als *plc_module* und das Dictionary mit den Informationen im Format *yaml* in die Datenbank gespielt. Die erstellten Methoden sind im Folgenden kurz beschrieben.

get_program Extrahiert alle Programme aus der Export-Datei und schreibt diese in das Dictionary *programs_dict*.

get_pid_header Erzeugt das Dictionary *pid_prg_dict*, welches die Programmaufrufe und die Namen der enthaltenen PID-Regler enthält.

get_clamp_info Extrahiert und speichert die Hardwareinformationen der SPS (*plc_info_dict*) sowie die Hard- und Softwareinformationen der verwendeten Klemmen (*plc_clamps_dict*).

get_tasks Informationen der verwendeten Taks werden der Export-Datei entnommen und in das Dictionary *dict_tasks* geschrieben.

get_program_info Unterteilt die Programme in *programs_dict* mit Hilfe der Methode *get_data_type* aus der Klasse *STParser* in Anwenderprogramme und PID-Regler. Die Methode gibt eine Liste der Programme (*main_loop_programs*) sowie der Regler (*program_list_pid*) zurück.

get_main_loop_files_st Speichert die Anwenderprogramme in das Dictionary *main_loop_files_dict*. Dafür wird die Liste *main_loop_programs* genutzt.

get_pid_info Extrahiert mit Hilfe der Liste *program_list_pid* die Parameter der PID-Regler und speichert sie im Dictionary *dict_pid*.

get_global_variables Extrahiert die globalen Variablen aus dem Steuerungsprogramm und speichert diese im Dictionary *plc_global_vars*.

Im Folgenden sind die im Dictionary *metadata_dict* enthaltenen Dictionaries aufgezählt, welche durch die oben beschriebenen Methoden erstellt werden.

programs_dict Enthält die in *CoDeSys 2.3* erstellten Programmbausteine. Der Schlüssel ist dabei der Programmname, das zugehörige Objekt das Anwenderprogramm. Neben Anwenderprogramme für den Steuerungsablauf werden in dem Dictionary bspw. auch Programme für PID-Regler gespeichert.

pid_prg_dict Alle in der Steuerung enthaltenen PID-Regler sind in diesem Dictionary gespeichert. Der Name des Reglers fungiert als Schlüssel, die Regelgröße als Objekt. Die PID-Regler können im Programm nur erfasst werden, wenn sie als Baustein vom Typ *Programm* in der Sprache FUP implementiert sind. Die Programmiersprache FUP wurde für diesen Bausteine *PID* verwendet, da die Handhabung dieser einfacher und anwenderfreundlicher als in der Sprache ST ist.

plc_info_dict Die Konfigurationseinstellungen der Steuerung sind in diesem Dictionary gespeichert. Dazu gehören Hardwarekonfigurationen und Informationen zu Kommunikationsschnittstellen, welche jedoch aktuell nicht weiter verwendet werden.

plc_clamps_dict Die an der SPS angeschlossenen Klemmen sind mit den zugehörigen Bezeichnungen der Klemmen sowie der vom Nutzer definierten Ein- und Ausgangssignale in diesem Dictionary genannt. Weitere Informationen zu den Klemmen, welche im Umfang dieses Programmes nicht benötigt werden, sind ebenfalls für eine spätere Nutzung vorhanden.

dict_tasks Die Konfigurationseinstellungen der einzelnen Tasks in CODESYS sind in diesem Dictionary enthalten. Dazu gehören unter anderem die Prioritäten der Tasks sowie die Intervalldauer der Aufrufe. Zu jeder Task wird ein Dictionary erstellt, welches die jeweiligen Informationen beinhaltet.

dict_pid Alle Parameter der verwendeten PID-Regler sind in in diesem Dictionary gespeichert. Zu jedem Regler gibt es ein eigenes Dictionary, in dem Werte wie *Kp* oder *Tn* eingetragen sind.

plc_global_vars Die globalen Variablen aus dem Steuerungsprogramm werden mit den gegebenen Werten abgespeichert. Falls keine Werte vorgegeben sind, werden die Variablen als *0* oder *FALSE* festgelegt, da dies die Initwerte bei CODESYS sind.

3.3.4. Klasse STParser

Die Funktionen der einzelnen Methoden zum Umwandeln der Programmiersprachen, die in der Klasse *STParser* gesammelt sind, werden in diesem Kapitel beschrieben. Dabei wird jedoch nicht mehr detailliert auf die Codierung eingegangen, da diese im Grundsatz auf der im Kapitel 3.3.2 erläuterten Funktionalität beruht. Die einzelnen Methoden werden in der Methode *parse* in der unten vorliegenden Reihenfolge aufgerufen. Aufgrund von Bedingungen,

die für spezielle Methoden erfüllt sein müssen, ist die Reihenfolge der Methodenaufrufe für ein erfolgreiches Übersetzen relevant. Teile des Codes werden durch die Methoden geändert, sofern dieser die Syntax der gesuchten Sequenz oder des Token enthält und an die nächste Methode weitergereicht. Somit wird der ST-Code Stück für Stück in Python übersetzt (siehe Abbildung 3.6 auf Seite 46). Der dabei weitergereichte Code ist als *hybrid_code* definiert. Im Anschluss werden die Operatoren wie in Tabelle 3.2 beschrieben übersetzt. Zur Wahrung der Übersichtlichkeit können die verwendeten Token und Sequenzen der Methoden im Quellcode A.2 im Anhang eingesehen werden.

get_data_type Extrahiert die Datentypen der verwendeten Variablen und speichert diese mit den Variablen im Dictionary *data_type_dict*.

get_start_values Initialwerte der Variablen werden in einer Zeichenkette gespeichert.

delete_comments Sucht nach Kommentaren in Anwendercode und löscht diese.

parse_vector_element Passt die Aufrufe von Vektorelementen an. Dies ist notwendig, da in ST das erste Element des Vektors in Python dem nullten Element entspricht.

get_timer_function Erzeugt ein Dictionary, welches die im Anwenderprogramm enthaltenen Timer und deren Einstellungen enthält.

get_timer_objects Extrahiert die Wartezeit der Timer und wandelt diese in Sekunden und Millisekunden um. Die für das Timerobjekt notwendigen Variablen (Aktitiverungsbit, Wartezeit und Periode) werden als aufrufbare Zeichenkette gespeichert.

find_timer Ersetzen der Timeraufrufe durch bestimmte Klassenaufrufe, da das Verhalten der Timer *TON*, *TOF* und *TP* mit Hilfe der definierten Klasse *TIMER* (Kapitel 3.3.6) beschrieben ist. Die Obekte werden im Skript *init_process_code* initialisiert und im Code *main_loop_code* ausgeführt.

delete_timer_def Die Instanziierung der Timer im Anwendercode wird gelöscht.

get_variables Die lokalen Variablen der Anwenderprogramme werden in dem Dictionary *plc_vars* mit den zugehörigen Werten gespeichert.

parse_rotate_right Wandelt die Funktion Rechtsrotation (*ROR*) aus ST mit Hilfe der Operatoren « und » um. Dabei muss der Datentyp der Variable in ST berücksichtigt und die Ausgabe mit Hilfe von *if*-Abfragen angepasst werden. Es können nur Variablen der Datentypen *BYTE*, *WORD* und *INT* konvertiert werden.

parse_rotate_left Wandelt die Funktion Linksrotation (*ROL*) aus ST mit Hilfe der Operatoren « und » um. Dabei muss der Datentyp der Variable in ST berücksichtigt und die Ausgabe mit Hilfe von *if*-Abfragen angepasst werden. Es können nur Variablen der Datentypen *BYTE*, *WORD* und *INT* konvertiert werden.

parse_limit Ersetzt die LIMIT-Funktion durch eine zusammengesetzte *min/max*-Überprüfung.

delete_REAL_TO_WORD Löscht die spezifische Umwandlung von Datentypen, welche in der *CoDeSys 2.3*-Umgebung zum Schreiben von Reglerausgängen auf Ausgangsklemmen benötigt wird.

parse_mux Ersetzt die MUX-Funktion durch eine *if*-Abfrage

delete_whitespaces Leerzeichen am Anfang der Zeilen werden gelöscht, damit es bei den Funktionen für die Einrückungen keinen Leerzeichen-Offset gibt.

parse_selection Wandelt die SEL-Funktion in eine *if*-Abfrage

parse_shift_right Übersetzt das bitweise Rechtsshiften (*SHR*) aus ST mit Hilfe der Operatoren *«* und *»*. Der Datentyp der Variable in ST wird mit Hilfe von *if*-Abfragen angepasst. Es können nur Variablen der Datentypen *BYTE*, *WORD*, *DWORD* und *INT* umgewandelt werden.

parse_shift_left Übersetzt das bitweise Linksshiften (*SHL*) aus ST mit Hilfe der Operatoren *«* und *»*. Der Datentyp der Variable in ST wird mit Hilfe von *if*-Abfragen angepasst. Es können nur Variablen der Datentypen *BYTE*, *WORD*, *DWORD* und *INT* umgewandelt werden.

parse_time Das übliche Zeitformat von ST wird in das Zeitformat *SECONDS.MILLISECONDS* gewandelt.

parse_case_row Speichert Zahlenreihen, die bei *CASE*-Abfragen benutzt werden, in Listen.

parse_case_listing Schreibt Aufzählungen aus *CASE*-Abfragen untereinander, sodass die Methode *parse_case* sie weiterverarbeiten kann.

parse_case Wandelt *CASE*-Abfragen in *if*- und *elif*-Abfragen.

parse_if Mittels Grammatik der Suchsequenz werden die *if*-Abfragen übersetzt.

parse_for Variablen aus *FOR*-Schleifen in Anwenderprogramm werden gespeichert. Anschließend wird die Syntax der Schleife angepasst. Dabei wird berücksichtigt, dass die Funktionalität der Schleifen unterschiedlich ist, weshalb Werte am Ende der Schleife modifiziert werden müssen.

parse_while Die Syntax der *WHILE*-Schleifen in ST wird durch die Python-Syntax ersetzt.

parse_repeat Ersetzt *REPEAT*-Funktionen durch *while not* Schleifen. *REPEAT*-Funktionen führen die Anweisungen innerhalb des *REPEAT*-Kommandos mindestens einmal aus, weshalb diese zum einmaligen Aufruf vor der *while not* Schleife eingefügt werden.

indent_for_while Sucht im Code nach *for*- und *while*-Schleifen und rückt deren Inhalt um vier Leerzeichen ein.

indent_if Sucht im Code nach *if*-Abfragen und rückt deren Inhalt um vier Leerzeichen ein.

Die Methode *delete_whitespaces* wird ein zweites Mal vor den Methoden, welche für die Einrückungen der *for*-, *while*- und *if*-Abfragen zuständig sind, aufgerufen. Die zur Umwandlung entwickelten Methoden decken nicht alle in ST vorhandenden Funktionalitäten ab, da dies den Rahmen einer Masterthesis sprengen würde. Es wurde bei der Auswahl der Funktionen berücksichtigt, dass vor allem die für die Steuerung von Wärmeübergabestationen relevanten und häufig verwendeten Sequenzen benutzt werden können.

3.3.5. Funktion create_init_main

Die Funktion *create_init_main* wird bei der Initialisierung des Simulationsprogramms aufgerufen. Sie verwendet das/die Anwenderprogramm/e in Python und die Informationen aus dem Dictionary *metadata_dict*, um die Skripte *python_init_code* und *python_process_code* zu erstellen. Außerdem wird das Dictionary *plc_global_vars* aus der Datenbank ausgelesen und lokal gespeichert.

Skript *python_init_code*

Das Skript *python_init_code* wird in der Simulationsumgebung verwendet, um die PID-Regler, die Regelung des Antriebsteuergeräts und die Timer zu initialisieren. Alle Timer- und Steueraufrufe aus dem Anwenderprogramm sind darauf referenziert. Die PID-Regler werden in der Funktion *write_controller_initialization* mit Hilfe des Dictionaries *pid_dict* als solche initialisiert. Für die Verwendung der Regelung muss jedem PID-Regler ein Regler nachgeschaltet sein, welcher das Ausgangssignal in vom Aktor lesbaren Grenzen hält. Dieser Regler vom Typ *drive_controller* wird ebenfalls in dieser Funktion erstellt. Die Funktion *write_timer_initialization* fügt den Code für die Initialisierung der/des Timer(s) ein.

Skript *python_process_code*

Das Anwenderprogramm, welches auf der SPS mit einer bestimmten Taktung aufgerufen wird, wird in *Jarvis* durch das Skript *python_process_code* simuliert. In *CoDeSys 2.3* können Regler und Anwenderprogramme als unterschiedliche *Tasks* definiert werden, welche eigene Aufrufintervalle und Prioritäten besitzen. Diese bestimmen, welche Programme zu welchem Zeitpunkt aufgerufen werden. Das Anwenderprogramm liest in einem Arbeitsschritt die

Signaleingänge an den Klemmen ein, durchläuft den vom Nutzer definierten Steuerungsalgorithmus und gibt die berechneten Ausgangssignale an die Klemmen weiter. In der Simulationsumgebung ist es aufgrund der Backend-Architektur noch nicht möglich, dass Skripte nur zu bestimmten Rechenschritten ausgeführt werden. Aufgrund dessen wurde die Taktung für das Erstellen der Skripts nicht berücksichtigt. Die Werte der Variablen werden nach jedem Rechenschritt in dem Dictionary *plc_vars* gespeichert, welches zu Beginn der Simulation die Werte der globalen und der lokalen Variablen des Programms besitzen. Vor jedem Aufruf des Skripts *python_process_code* werden diese aus dem Dictionary in die Skriptumgebung geladen.

Die Funktion *get_clamp_connections* erstellt anhand der Informationen zu den Klemmen aus dem Dictionary *plc_clamps_dict* die Funktionsaufrufe zum Einlesen und Ausgeben der Signale. Eine Liste der *Tasks* wird mit den zugehörigen Prioritäten durch die Funktion *get_priority_list* erstellt. Anhand dieser Liste werden anschließend die unterschiedlichen Anwenderprogramme und Regler in der vorgegebenen Reihenfolge in das Skript durch die Funktion *write_main_loop_order* eingefügt. Die Aufrufe der PID-Regler werden unter Zuhilfenahme der Daten des Dictionaries *pid_prg_info* erstellt. Zuletzt wird die Bibliothek *math* in das Skript eingefügt. Mit Hilfe dieser Funktion können dem Anwenderprogramm durch den Benutzer Codezeilen in der Sprache Python hinzugefügt werden.

3.3.6. Klasse TIMER

Die Timeraufrufe aus dem Anwenderprogramm der Entwicklungsumgebung *CoDeSys 2.3* werden mit Hilfe der Funktionen der Klasse *TIMER* dargestellt. Neben der Funktion für die Initalisierung der Timer (*__init__*) gibt es die Funktionen *TON*, *TOF* und *TP*. Jede dieser Funktionen simuliert die in *CoDeSys 2.3* verwendbaren Timer. Dabei kann durch die Endung *.Q* und *.ET* von dem/der Anwender/in bestimmt werden, welcher Wert zurückgegeben werden soll. Es kann entweder der Ablauf der vorgegebenen Zeit nach Eintritt eines Ereignisses (*.Q*) oder die abgelaufene Zeit seit dem Timeraufruf *.ET* ausgegeben werden. Genauere Informationen zur Nutzung der Timer können in [21] eingesehen werden.

3.4. Nomenklatur und Einschränkungen

Die Unterschiede in der Programmarchitektur (Backend) der Entwicklungsumgebung *CoDeSys 2.3* und der Simulationsumgebung *Jarvis* erfordern, dass bei dem Code des Anwenderprogramms eine Namenskonvention eingehalten wird. Die wesentliche Voraussetzung bei der Verwendung des Importers und des Parsers ist, dass die Bezeichner (Namen) der einzelnen Komponenten in der Entwicklungsumgebung und im Simulationsprogramm übereinstimmen.

Bei den Temperatursensoren muss außerdem zwischen Sensoren im Speicher und den übrigen Sensoren vom Typ *Temperature_Sensor* unterschieden werden. Letztere benötigen für die korrekte Zuweisung im Simulationsprogramm den Zusatz *_clamp* als Suffix des Komponentennamens. Die Sensoren im Speicher müssen mit dem Präfix *storage_* versehen werden. Des Weiteren muss bei der Erstellung des Steuerungsprogramms darauf geachtet werden, dass PID-Regler vom Typ *Programm* nur verwendet werden können, wenn sie als Task in das Anwenderprogramm eingebunden sind und zyklisch aufgerufen werden. Ein Einbinden des Reglers in den ST-Code ist im aktuellen Entwicklungsstand des Parsers nicht möglich. Bei der Verwendung von Funktionen im ST-Code sollte beachtet werden, dass nur eine bestimmte Auswahl der Funktionen (siehe Kapitel 3.3.4) übersetzt werden kann.
Bzgl. der Bezeichnungen der Ausgangssignale der Regelung ist darauf zu achten, dass diese im Anwenderprogramm und bei der Ausgangsklemmen übereinstimmen und die verwendete Variable im Anwenderprogramm den Suffix “_” besitzt.

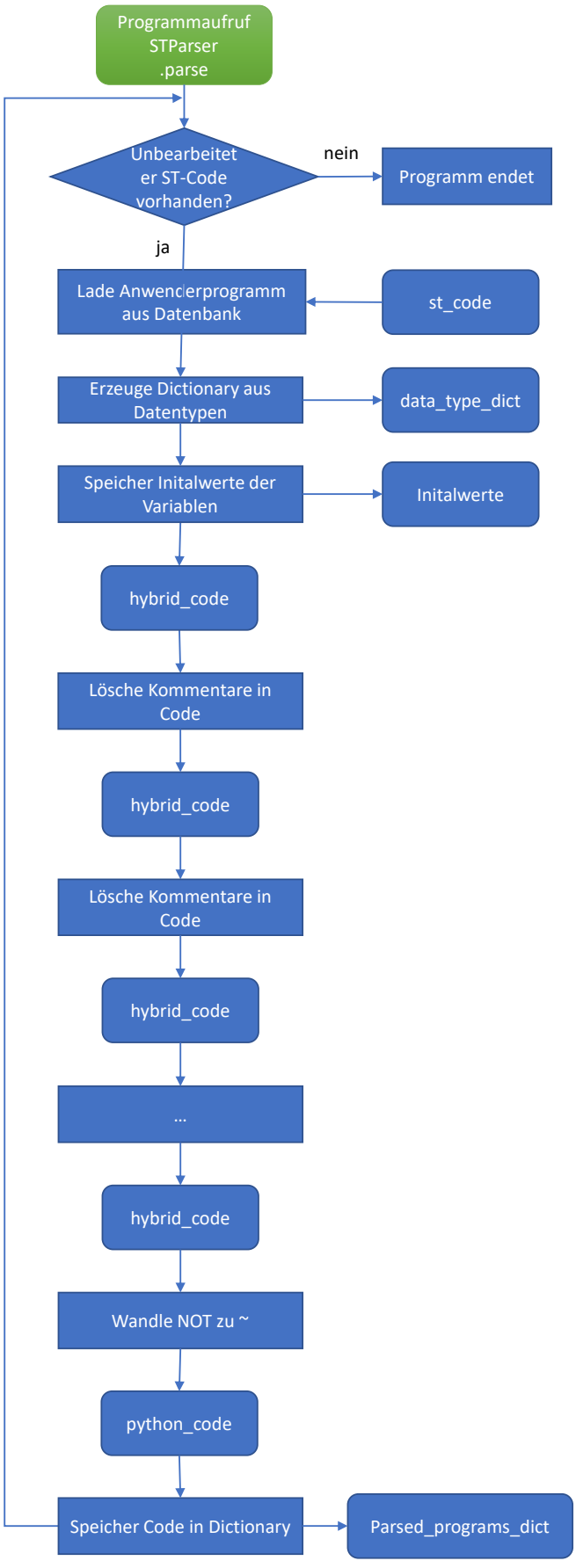


Abbildung 3.6.: Programmaublaufplan *parse*

4. Hardwaretestumgebung

Das folgende Kapitel befasst sich mit der Umsetzung der iWÜST mit TWW als Hardwaremodell. Einleitend wird der Nutzen eines solchen Modells erläutert. Im Kapitel 4.2 werden die entsprechenden Randbedingungen dafür festgelegt. Außerdem werden die Komponenten der realen Anlage und deren Umsetzung als Modell beschrieben. Zuletzt wird die Fertigung und Montage der Modellkomponenten erklärt.

4.1. Nutzen

Die hier vorgestellte SPS-basierte Testumgebung wurde erstellt, um die Funktionaltät des Regelalgorithmus einer iWÜST zu verifizieren und mögliche Regelstrategien zu veranschaulichen. Die im Regelalgorithmus berücksichtigten Regelszenarien sind in den Tabellen 2.1 und 2.2 aufgeführt. Eine Funktion des Algorithmus ist bspw., dass bei einer erhöhten Rücklauftemperatur einer TWW im Wärmenetz der primärseitige Durchfluss mit Hilfe eines Ventils angepasst und so die Rücklauftemperatur reduziert wird (siehe Szenario *Rücklauftemperatur* in Tabelle 2.1). In der Hardwaretestumgebung sollen die Reaktionen der drei Ventile und der Pumpe auf die Veränderungen der Sensor- und Messdaten dargestellt werden.

Wie bei der real umgesetzten iWÜST berechnet ein Algorithmus auf einer SPS die notwendigen Stellgrößen der einzelnen Aktoren (Ventile und Pumpe) unter Berücksichtigung der maßgeblichen Umgebungsparameter. Die notwendigen Signale für die Regelung stammen in der realen Umsetzung von einem Messgerät, mehreren Sensoren, schaltenden Temperatur- und Druckbegrenzer und der externen Leitwarte und werden über die Klemmen der SPS, via serieller Schnittstelle und Ethernetverbindung an die SPS gesendet. In der Modellvariante können diese Parameter individuell von außen vorgegeben bzw. verändert werden. Mit der Testumgebung sollen Reaktionen der Aktoren auf bestimmte Signaländerungen von Sensoren deutlich gemacht werden.

Für ein realitätsnahes Modell ist darauf geachtet worden, dass die Komponenten der SPS sowie die Ein- und Ausgangssignale denen einer realen Anlage entsprechen. Der Regelalgorithmus wird in der Sprache ST geschrieben und anschließend über den entwickelten Parser in ein für die Simulationsumgebung *Jarvis* lesbares Format gewandelt.

So lassen sich die SPS-Regelalgorithmen zusätzlich in einem Softwaremodell der iWÜST testen. Die Umsetzung des entwickelten Algorithmus der iWÜST folgt in Kapitel 6. Die Softwaretestumgebung folgt in Kapitel 5.

4.2. Randbedingungen

Grundsätzlich wird bei der Hardwaretestumgebung auf die Verwendung sämtlicher hydraulischer Komponenten verzichtet. Somit ist kein Wärmeträgermedium vorhanden. Es werden weder Wärmeerzeuger, -verbraucher oder -übertrager eingesetzt. Zusätzlich wurden bei den Vorüberlegungen zur Modellbildung die folgenden Randbedingungen für die Testumgebung bestimmt:

SPS Es soll eine WAGO-SPS der Modellreihe 880-750 verwendet werden, da diese auch für die umgesetzte Anlage der iWÜST im realen Wärmenetz genutzt wird.

Klemmenbelegung und Signale Die zur Übermittlung der Sensor- und Aktorsignale verwendeten Klemmen sollen denen entsprechen, die bei der realen iWÜST verwendet werden. Dementsprechend wird gefordert, dass die Signale, die von den im Modell verwendeten Sensoren generiert werden, die gleichen Signalstärken und -formate wie in der Realtiät besitzen.

Wahrnehmung der Ausgangssignale Für die Validierung und Veranschaulichung des Steuerungsalgorithmus sollen die Ausgangssignale visuell dargestellt werden, sodass die Auswirkungen von Veränderungen der Umgebungs- und Regelungsparameter leicht erkannt werden können.

Mobilität Die Hardwaretestumgebung soll ohne großen Aufwand auf- und abgebaut werden können. Eine handliche Größe soll einen einfachen Transport ermöglichen.

Erscheinung Die Testumgebung soll eine ansprechende äußere Form besitzen.

4.3. Steuerungstechnischer Aufbau der realen iWÜST

Zur Erfassung des aktuellen Zustands des Systems (iWÜST) sind eine Vielzahl von Druck- und Temperatursensoren sowie ein Wärmemengenzähler in der realen iWÜST verbaut. Ein Drei-Wege-Ventil, eine Pumpe und zwei Durchgangsventile sorgen dafür, dass die geforderten Systemtemperaturen sowie der gewünschte Speicherfüllstand eingestellt werden können. Anhand des R&I (Abbildung 2.8 auf Seite 12) kann der schematische Aufbau mit den Bauteilen eingesehen werden. Die im System enthaltenen steuerungstechnischen Komponenten

Tabelle 4.1.: Steuerungstechnische Komponenten der iWÜST

Komponente	Anzahl	Wertebereich (Anwenderprogramm)	Dateiformat	Schnittstelle (SPS)
Temperatursensor	12	−2000..8500	INT	Analogeingang für PT1000
Druckbegrenzer	1	TRUE/FALSE	BOOL	Digitaleingang DC 24 V
Temperaturbegrenzer	1	TRUE/FALSE	BOOL	Digitaleingang DC24 V
Wärmemengenzähler	1	ARRAY mit Werten	REAL	serielle Schnittstelle
Pumpe	1	0..32 768	WORD	Analogausgang DC 0..10 V
3-Wege-Ventil	1	0 / 32 767	WORD	Analogausgang DC 4..20 mA
Durchgangsventil	2	0..32 767	WORD	Analogausgang DC 4..20 mA

(Sensoren und Aktoren) sind in Tabelle 4.1 dargestellt. Die verwendete SPS wurde in Kapitel 2.3.1 beschrieben.

4.3.1. Funktionsbeschreibung und Bezeichnung der steuerungstechnischen Komponenten

Die Bezeichnung der Komponenten setzt sich aus einem Kürzel der Komponente und einer Zahl zusammen. Die Hunderterstelle der Zahl gibt an, in welchem Teilbereich der iWÜST sie sich befindet (vgl. Abbildung 2.8). Die Zehnerstelle spezifiziert einen Bereich in dem Teilbereich (z.B. den Speicher). Die letzte Ziffer definiert die bestimmte Komponente in dem Teilbereich. Die Komponententypen entsprechen denen, die in der realen Anlage verwendet werden.

Temperatursensor Die PT1000-Temperatursensoren sind ohmsche Widerstände, deren Wert sich linear mit der den Sensor umgebenden Temperatur ändert. Es handelt sich hierbei um Tauchfühler, die in die hydraulischen Komponenten eingebaut sind. Bei einer Umgebungstemperatur von $0\,^{\circ}\mathrm{C}$ liegt der Widerstandswert bei $1000\,\Omega$. Die Temperatursensoren werden mit dem Kürzeln *TIRC*, *TIR* und *TC* nach *DIN19227-1* und der beschriebenen Zahl als Suffix angegeben.

Überdruckbegrenzer/Übertemperaturbegrenzer Die im System verbauten Begrenzer fungieren wie Schalter, die bei Überschreiten eines bestimmten Grenzwerts den Stromfluss unterbrechen. Dadurch wird die Verbindung der beiden Klemmanschlüsse unterbrochen, was vom Steuerungssystem detektiert werden kann. Die Begrenzer werden mit *TZ+* (Druck) und *TS+* (Temperatur) und einer Zahl definiert.

Wärmemengenzähler Der Wärmemengenzähler ist in der Abbildung 2.8 unter *WMZ001* eingezeichnet. Es handelt sich dabei um ein Messgerät, das die Temperaturen im Vor- und Rücklauf des Systems mit Temperatursensoren erfasst. Außerdem wird der Volumenstrom im Vor- oder Rücklauf gemessen. Mit diesen Messwerten berechnet der Wärmemengenzähler dann die übertragene Wärmemenge $\dot{Q}$ (vgl. Formel 2.3). Im Abstand von 30 Sekunden werden diese Ergebnisse dann von dem verwendeten Wärmemengenzähler über den Feldbus *M-Bus* an die SPS geschickt. Mit Hilfe eines Konverters wird das Signal in das Protokoll *Modbus* umgewandelt und kann dann von einer Klemme an der SPS ausgelesen werden.

Pumpe Die regelbare Pumpe der iWÜST wird als *P401* bezeichnet. Sie fördert, abhängig von der anliegenden Signalspannung, eine bestimmte Menge des Wärmeträgermediums. Die Pumpenleistung ist linear abhängig von der anliegenden Steuerspannung, die im Bereich von 0..10 V liegt.

Durchgangsventil Das System enthält zwei Durchgangsventile. Die Ventilstellung der Ventile ist linear abhängig vom anliegenden Signalstrom, der im Wertebereich von 4..20 mA liegt. Bei maximalem Signalstrom ist das Ventil komplett geöffnet, bei minimalem Strom komplett geschlossen. Die Ventile besitzen das Kürzel *MV* (Motorventil). Das Ventil *MV101* reguliert den primärseitigen Durchfluss des Wärmeübertragers zur Versorgung der Heizkreise, wohingegen die Stellung des Ventils *MV301* ausschlaggebend für das Be- und Entladen des Speichers ist.

3-Wege-Ventil Das 3-Wege-Ventil (*MV302*) wird über einen Signalstrom angesteuert. Bei einem Eingangssignal von 4 mA fließt das Wärmeträgermedium aufgrund der Ventilstellung in den Speicher, bei 20 mA in den Netzrücklauf. Dadurch kann bei hohen Rücklauftemperaturen aus dem Wärmeübertrager der TWW die enthaltene Wärmeenergie genutzt werden, um den Speicher warm zu halten.

4.3.2. Konzeptionierung

Für die Hardwaretestumgebung wurden verschiedene Bauteile benötigt, damit die eigentlichen Systemkomponenten der realen iWÜST nachgebildet werden können. Dabei wurden die oben gestellten Randbedingungen an das Modell berücksichtigt. Es wird in den folgenden

Abschnitten erklärt, wie die Komponenten aus Kapitel 4.3.1 in der Hardwaretestumgebung umgesetzt wurden.

Temperatursensoren

Einige Regelszenarien der iWÜST werden durch das Über- oder Unterschreiten einer Grenztemperatur eines bestimmten Temperatursensors ausgelöst bzw. aufgehoben. Des Weiteren wird der Füllstand anhand der Temperaturen im Speicher berechnet. Um dies in einem Modell darstellen zu können ist es notwendig, dass die Bauteile, welche die Temperatursensoren beschreiben, veränderbar sind. Dies wurde durch den Einsatz von Potentiometern erreicht. Potentiometer sind elektrische Widerstände, deren Wert durch einen Drehknopf linear verändert werden kann. Schließt man diese anstatt PT1000-Sensoren an die Klemmen der SPS, so kann man die in der Steuerung eingelesene Temperatur durch Drehen verändern, da die Steuerung lediglich den Widerstandswert erfasst. Damit die Temperatur in $0{,}1\,^{\circ}\mathrm{C}$-Schritten in einem realistischen Bereich eingestellt werden kann, wurde zu jedem Potentiometer ein Vorwiderstand von 806 Ω in Reihe geschaltet, welcher einen Temperaturoffset von $-50\,^{\circ}\mathrm{C}$ bewirkt. Die 10-Gang-Potentiometer können Widerstandswerte von 0-1000Ω erreichen. Somit kann durch diesen Aufbau eine Temperatur im Bereich von $-50\,^{\circ}\mathrm{C}$ bis $220\,^{\circ}\mathrm{C}$ eingestellt werden. Ohne den Vorwiderstand würde die Temperatur in einem Bereich von $-200\,^{\circ}\mathrm{C}$ bis $70\,^{\circ}\mathrm{C}$ eingestellt werden können.

Druck- und Temperaturbegrenzer

Die Temperaturbegrenzer wurden in dem Modell durch Ein-/Aus-Schalter ersetzt. Durch Betätigen dieser Schalter ist der Stromkreis an den Klemmeingängen nicht mehr geschlossen, was von der Steuerung detektiert wird und dem Verhalten der Begrenzer beim Überschreiten des Grenzwert entspricht.

Wärmemengenzähler

Der Wärmemengenzähler ist ein eigenständiges Gerät mit eingebautem Mikrocontroller und eigener Sensorik. Das Ersetzen dieses Geräts mit einem äquivalenten Bauteil hätte den Rahmen dieser Arbeit gesprengt. Deshalb wurde vereinfacht im Anwenderprogramm der SPS ein Code implementiert, welcher Leistungs- und Volumenstromwerte mit Hilfe von Temperaturen und Ventilstellungen berechnet.
Die als Wärmeleistung (*PWMZ001*) vom Wärmemengenzähler simulierten Werte werden mit Hilfe der Netztemperatur, der Temperaturen der beiden Rückläufe und den Ventilstellungen berechnet. Diese Berechnung wird mit Hilfe eines Timers alle 30 Sekunden durchgeführt,

was der Taktung eines handelsüblichen Wärmemengenzählers entspricht. Die Öffnung der Ventilstellung (*Ventilstellung_MV101/Ventilstellung_MV301*) wird von null bis eins skaliert und mit dem maximalen Volumenstrom des jeweiligen Strangs multipliziert. Die maximalen Volumenströme ($\dot{V}$_MV301_max=1,7 kg/s, $\dot{V}$_MV101_max=0,7 kg/s) entsprechen denen aus der Simulation (Kapitel 5). Dadurch wird die fiktive Leistung mit Hilfe der Volumenströme $\dot{V}$_MV101 und $\dot{V}$_MV301 nach Gleichung 2.3 wie folgt berechnet.

$$\dot{V}\_MV101 = Ventilstellung\_MV101 \cdot 0.7 \tag{4.1}$$

$$\dot{V}\_MV101 = Ventilstellung\_MV301 \cdot 1.7 \tag{4.2}$$

$$\begin{aligned} \dot{Q} = (\dot{V}\_MV101 \cdot (T\_Netz\_soll - TIRC103) + \\ \dot{V}\_MV301 \cdot (T\_Netz\_soll - TIRC305)) \cdot c_p \end{aligned} \tag{4.3}$$

Die von der Außentemperatur abhängige Netztemperatur (*T_Netz_soll*) wird durch das von einem Timer gesteuerte wiederkehrende Aufrufen des Codes in Quellcode 4.1 simuliert.

Quellcode 4.1: Quellcode der Netztemperaturberechnung

```
1 IF T_aussen > T_aussen_max THEN
2 T_delta := -0.01;
3 ELSIF T_aussen < T_aussen_min THEN
4 T_delta := 0.01;
5 END_IF;
6 T_Netz_soll := T_Netz_start - (T_aussen/2);
```

Im Abstand von 30 Sekunden steigt bzw. sinkt die Außentemperatur *T_aussen* um 0,1 °C in den vorgegebenen Grenzen von −10 °C bis 30 °C. Dadurch wird in zeitlich erfassbaren Abschnitten getestet, ob der Steuerungsalgorithmus bei unterschiedlichen Außentemperaturen richtig reagiert. Sprünge in der Außentemperatur sind nicht vorgesehen, da diese nicht in der Realität vorkommen. Die Temperatur *T_Netz_start* beträgt 85 °C. In der Simulations des Modells (Kapitel 5) wird die Netztemperatur als Sollwert vorgegeben, welcher durch einen simulierten Wärmeerzeuger eingestellt wird. In der Hardwaretestumgebung entspricht der Istwert der Temperatur (*T_Netz_ist*) dem Sollwert *T_Netz_soll*. Abbildung 4.1 stellt den simulierten Temperaturverlauf der Außentemperatur sowie der davon abhängigen Netztemperatur für einen Zeitraum von 13 Stunden dar.

Pumpe

Die Pumpenleistung, welche abhängig von der anliegenden Steuerspannung ist, wird durch die Helligkeit einer blauen Leuchtdiode (LED) visualisiert. Leuchtet diese hell, so liegt eine

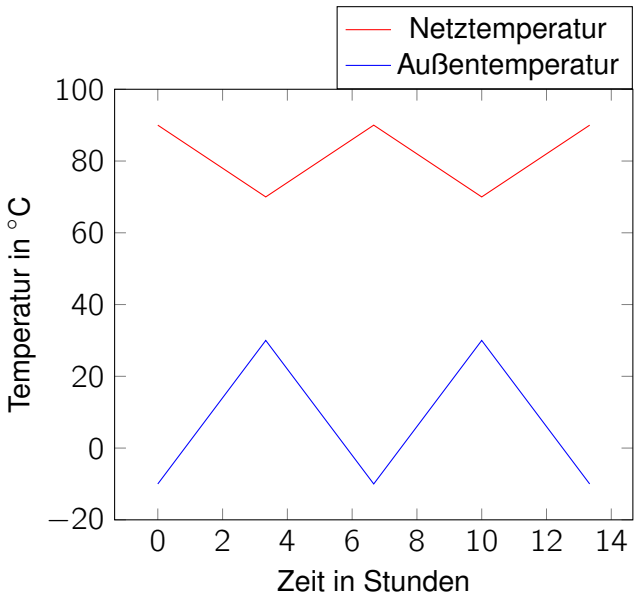


Abbildung 4.1.: Verlauf der Außen- und Netztemperatur

hohe Spannung an. In der Realität würde die Pumpe also einen großen Volumenstrom fördern. Leuchtet die LED schwach oder gar nicht, so entspricht das einer geringen bzw. gar keiner Fördermenge.

Ventile

Die Ventilstellungen des Mischventils und der Durchgangsventile werden durch analoge Amperemeter in einem Bereich von 4..20 mA dargestellt. Die Stromstärke wird von der Regelung der SPS mit Hilfe eines PID-Reglers berechnet und über die Klemmen an die Amperemeter weitergegeben. Die praktische Umsetzung der Anzeige ist im folgenden Kapitel 4.4 beschrieben.

4.4. Umsetzung

Die Hardwaretestumgebung wurde in der Werkstatt des Energiecampus der HAW in Hamburg-Bergedorf selbstständig erstellt. Die für die Umsetzung benötigten Bauteile können in der Abbildung A.4 im Anhang eingesehen werden. Generell besteht die Testumgebung aus einem Schaltkasten und einer Schalttafel.

4.4.1. Schaltkasten

Im Schaltkasten ist die SPS mit den Klemmanschlüssen sowie dessen Netzteil auf einer Hutschiene angebracht. Die SPS-Klemmanschlüsse sind durch blaue Leitungen mit Durchgangs- und Trennklemmen auf einer zweiten Hutschiene verbunden. In den Trennklemmen sind die Vorwiderstände der Potentiometer durch eingesteckte Bauelemente-Stecker integriert. Die Durchgangs- und Trennklemmen sind durch 36 Einzelkabel mit der Schalttafel elektrisch verbunden, wobei diese Verbindungen über sogenannte Bananenstecker und -buchsen auf der Schalttafel ausgeführt sind. Dies ermöglicht eine schnelle Trennung der beiden Bauteile, sodass die Testumgebung schnell und räumlich flexibel auf- und abgebaut werden kann. Es erleichtert außerdem den Transport; weiterhin können die anliegenden Spannungen und Ströme können für eine Kontrolle mit dem Multimeter besser abgegriffen werden. Dafür sind zwölf Mal jeweils drei Kabel aus einer der Öffnungen des Schaltkastens geführt.
Die drei Kabel können an den Farben der Stecker unterschieden werden. Auf der Schalttafel gibt es zwölf Reihen von Buchsen in den Farben der Stecker. Bei Anordnung der Schalttafel und des Kastens wie in Abbildung 4.2 müssen die obersten drei Stecker in die farblich passenden Buchsen der obersten Zeile gesteckt werden. Mit den folgenden Steckverbindungen wird nach gleichem Verfahren vorgegangen, bis die auf Abbildung 4.2 unten aus dem Schaltkasten austretenden Kabel in die letzte Reihe gesteckt werden. Abbildung A.5 im Anhang zeigt den elektrischen Schaltplan der Hardwaretestumgebung. Anhand der Abbildungen A.6, A.7 und A.8 im Anhang können die Bezeichnungen des Schaltplans nachvollzogen werden.

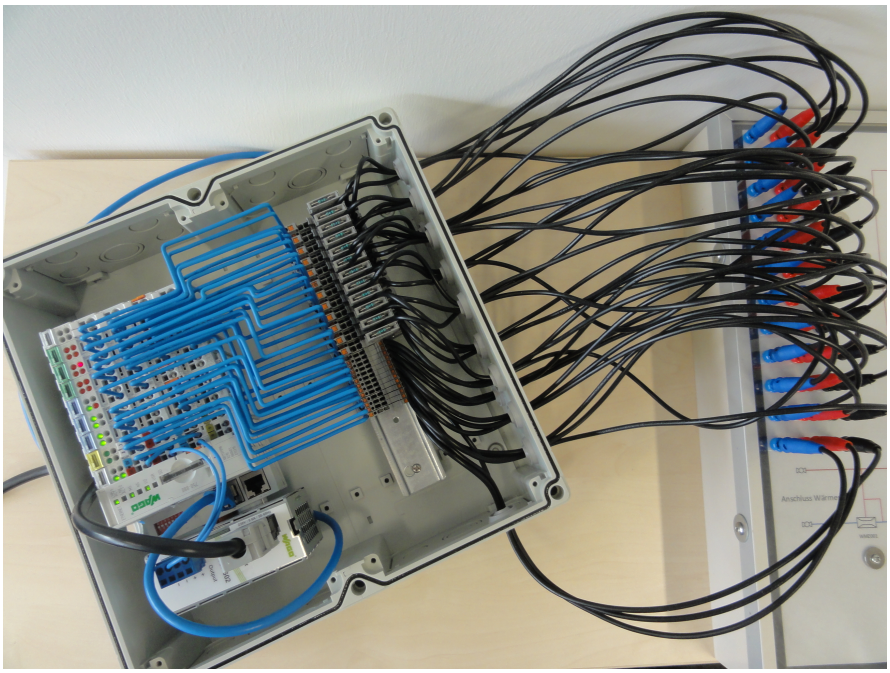


Abbildung 4.2.: Schaltkasten und Verbindungen zur Schalttafel

4.4.2. Schalttafel

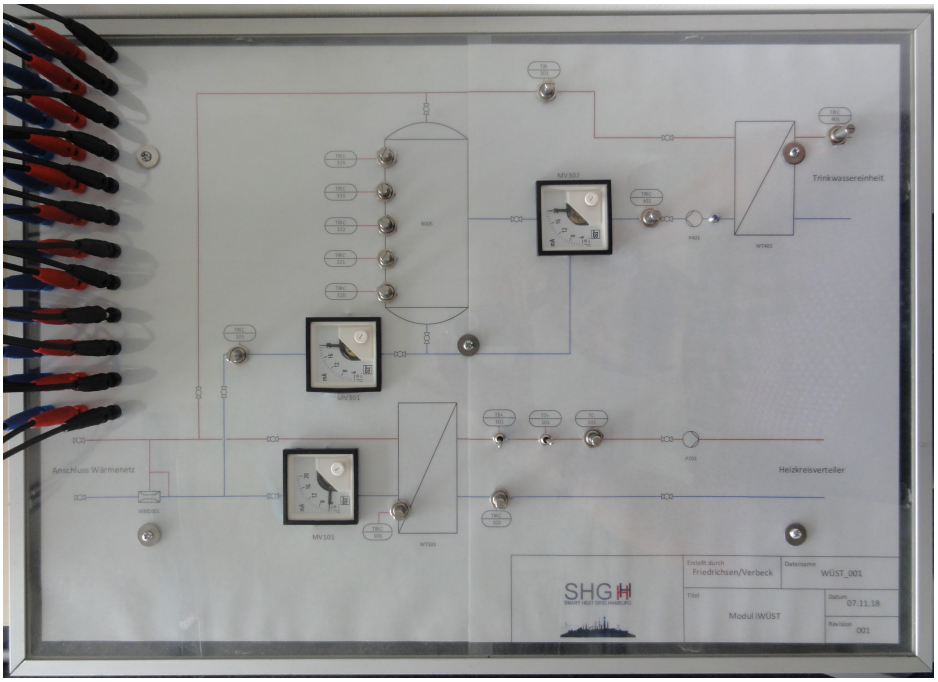


Abbildung 4.3.: Schalttafel der Hardwaretestumgebung

Bei der Schalttafel handelt es sich um eine Acrylglasplatte, in die die beschriebenen Bauteile der Sensorik und Aktorik aus Kapitel 4.3.2 eingebaut sind. Die Größe der Schalttafel ist ein Kompromiss aus geforderter Kompaktheit, Übersichtlichkeit und Berücksichtigung des Platzbedarfs der benötigten Bauteile. Hinter der Acrylglasplatte befindet sich ein laminierter R&I-Plan der iWÜST. Die Bauteile wurden an den Stellen platziert, wo sie im R&I-Plan zu finden sind (siehe Abbildung 4.3). Die dafür benötigten Öffnungen in der Acrylglasplatte wurden in der Maschinenhalle des Campus der HAW in Hamburg-Bergedorf mit einer Wasserstrahlschneidemaschine geschnitten. Die Zeichnung mit den zugehörigen Maßen ist in der Abbildung A.9 im Anhang einzusehen.

Das Mischventil *MV302* ist so eingebaut, dass der Zeiger des Messgeräts die von der Steuerung vorgegebene Fließrichtung des fiktiven Wärmeträgermediums aus dem Rücklauf des Wärmeübertragers der TWW anzeigt. Die Zeiger der Amperemeter, die die Durchgangsventile *MV101* und *MV301* beschreiben, sind bei komplett geöffnetem Ventil Plan mit den im R&I vorhandenen hydraulischen Rohrverbindungen. Bei geschlossenem Ventil steht der Zeiger zu den Linien in einem Winkel von 90°. Auf dem Foto in Abbildung 4.3 ist dies zu erkennen. Zum Zeitpunkt der Aufnahme war das Mischventil *MV302* so eingestellt, dass das Wärmeträgermedium in den mittleren Stutzen des Wärmespeichers eingespeist werden würde. Das Ventil *MV301* ist komplett geöffnet, wohingegen *MV101* geschlossen ist. Würde

die Steuerung ein Stellsignal von 12 mA vorgeben, so würde der Zeiger 12 mA anzeigen. Dies bedeutete, dass das Ventil halb geöffnet wäre.
Die Bananenbuchsen sind über angelötete Leitungen mit den Bauteilen der Schalttafel verbunden. Abbildung 4.4 zeigt die Rückseite der Tafel, auf der die Bauteile und die Leitungen zu sehen sind.

Im Bereich der Ecken und im Mittelpunkt der Platte sind Holzstäbe mit einer Länge von 6,3 cm als zusätzliche Stützen angebracht. Diese sorgen dafür, dass die Gehäuse der Amperemeter nicht auf dem Boden aufliegen und somit das Gewicht der Platte halten. Zur Verbesserung der äußeren Erscheinung und der Transportmöglichkeit wurde eine Kiste aus Holz angefertigt, in die der beschriebene Aufbau hineingestellt werden kann. Dieser ist dann mit Aluminiumschienen an der Kiste befestigt. Dadurch sind die Verbindungen der Buchsen zu den Bauteilen vor äußeren Beschädigungen geschützt.

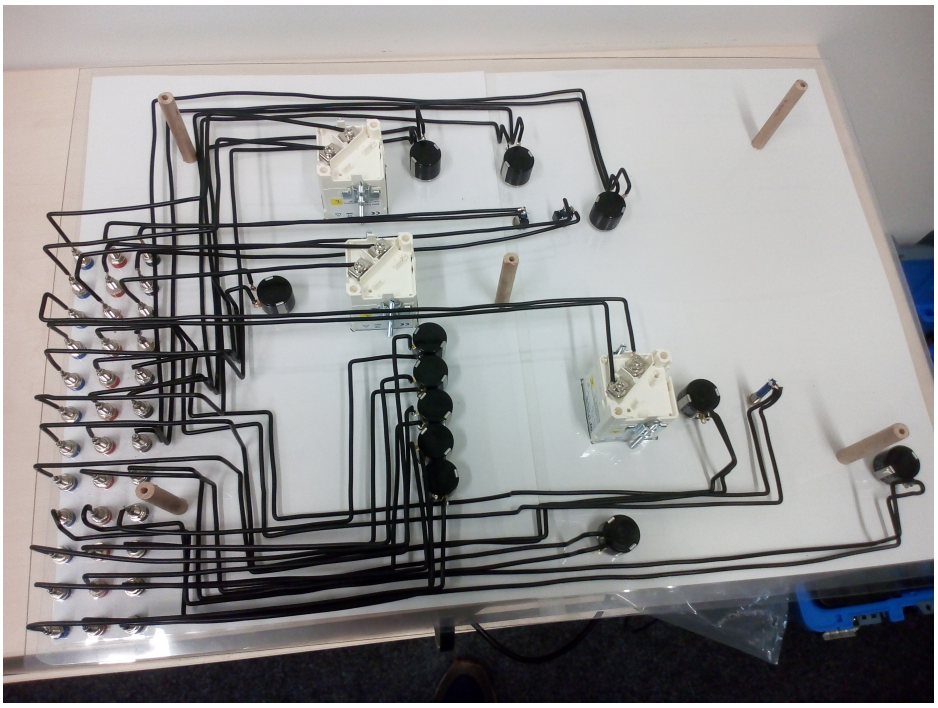


Abbildung 4.4.: Rückseite der Schalttafel

5. Simulationsmodell

Die Simulation der iWÜST mit TWW wurde mit der im Kapitel 2.6 beschriebenen Simulationsumgebung *Jarvis* umgesetzt. Im Folgenden wird die Modellbildung an Hand des hydraulischen Aufbaus der Komponenten und deren Parametrierung beschrieben. Hierbei wird auch darauf eingegangen, welche Komponenten als Ersatz für Bauteile entwickelt wurden, die nicht in der Simulationsbibliothek vorhanden sind.
Das Modell der iWÜST wurde zunächst in der Variante *druckbehaftet* erstellt. Dies war das erste "größere" Modell, das mit dem kurz zuvor fertiggestellten *Druckmonolith* erstellt wurde. Da der Solver bei der Berechnung jedoch keine richtigen Ergebnisse generierte, wurde das Modell der iWÜST in der Variante *drucklos* modelliert. Dieses Modell wurde für die weiteren Arbeitsschritte verwendet. Der Verzicht auf druckbehaftete Komponenten hatte zur Folge, dass die in der realen iWÜST vorhandenen Ventile im Modell durch Pumpen ersetzt werden mussten. Dies wird im folgenden Kapitel 5.2 näher beschrieben. Auf die Validierung der Regelbeschreibung der iWÜST wirkt sich dies nicht aus, da die Reaktion der Aktoren auf veränderte Messwerte in beiden Fällen gleich ist. Für die Anwendung des Modells in anderen Simulationen sollte jedoch die Machbarkeit eines druckbehafteten Berechnungsansatzes nochmals geprüft und gegebenenfalls umgesetzt werden.

5.1. Hydraulischer Aufbau

Das Modell besteht aus den drei geschlossenen Wasserkreisläufen *Wärmenetz*, *Heizkreis* und *TWW*, welche in den folgenden Abschnitten im Bezug auf die hydraulischen Komponenten definiert werden.

Wasserkreislauf Wärmenetz

Im Kreislauf *Wärmenetz* wird die Zirkulation des Wärmeträgermediums Wasser simuliert. Das Wärmenetz wird durch einen Wärmeerzeuger dargestellt. Vier Pumpen (*P401*, *MV101*, *MV301*, *MV302*) sorgen dafür, dass die gewünschten Systemzustände erreicht werden können. Zwei Wärmespeicher, die durch ein T-Stück verbunden sind, simulieren einen Wärmespeicher mit mittlerem Ladestutzen. Des Weiteren enthält der Kreis zwei Wärmeübertrager, welche die

anderen beiden Kreisläufe mit Wärmeenergie versorgen. Rohrleitungen verbinden einzelne Komponenten miteinander. In Abbildung 5.1 ist dies anhand der Darstellung aus dem *Frontend* der Simulationsumgebung gezeigt.

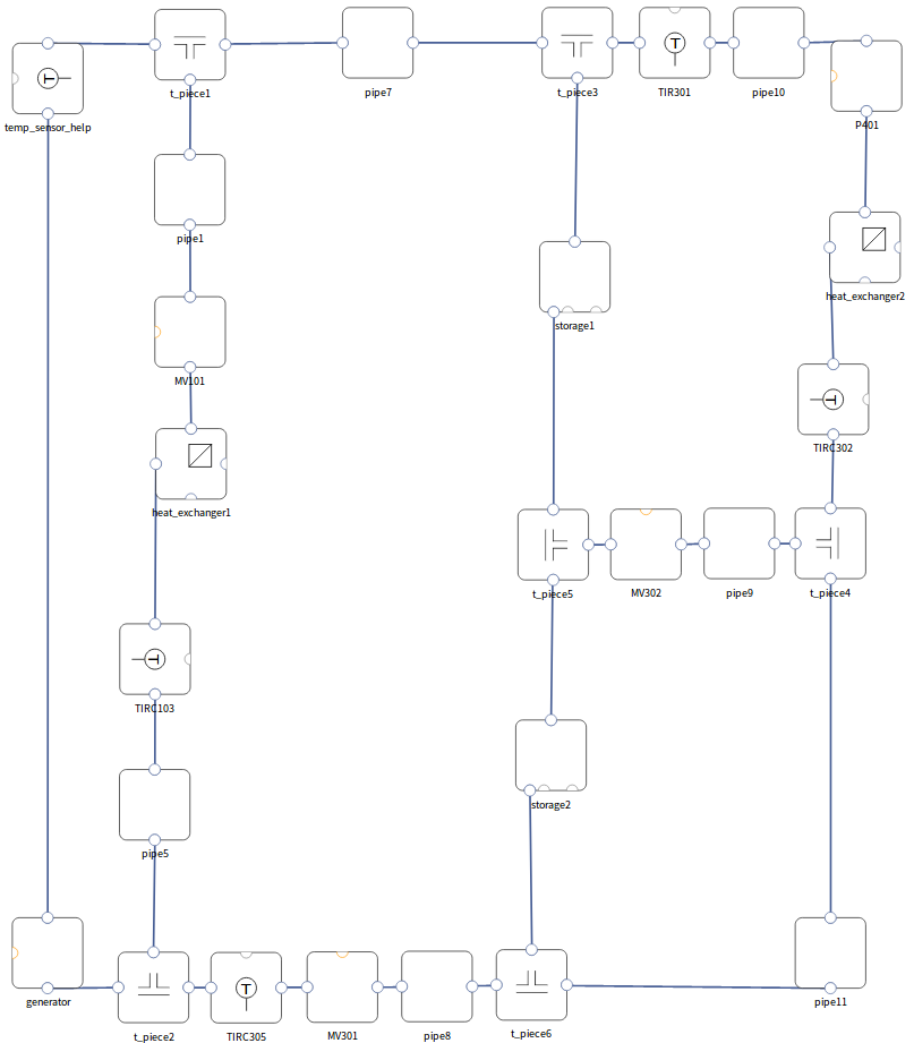


Abbildung 5.1.: Wasserkreislauf Wärmenetz

Wasserkreislauf Heizkreis

Die Wärmeabnahme bzw. Last wird durch den Kreislauf *Heizkreis* simuliert. Dieser besteht aus einem Wärmeübertrager, einer Pumpe, einer Last und zwei Rohrleitungen (siehe Abbildung 5.2). Die Last reduziert durch ihre Leistungsabnahme die Wassertemperatur. Für die Regelung der iWÜST sind lediglich die Temperaturen am Wärmeübertrager sowie der Temperatur-

und Druckbegrenzer interessant, weshalb der/die Heizkreis(e) vereinfacht durch die Last dargestellt werden können.

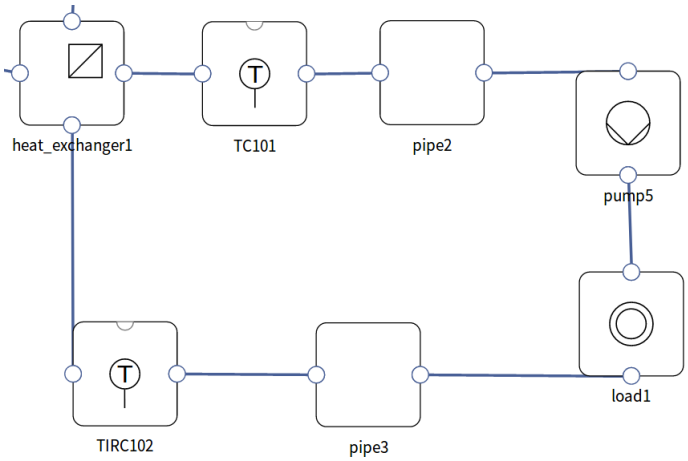


Abbildung 5.2.: Wasserkreislauf Heizkreis

Wasserkreislauf TWW

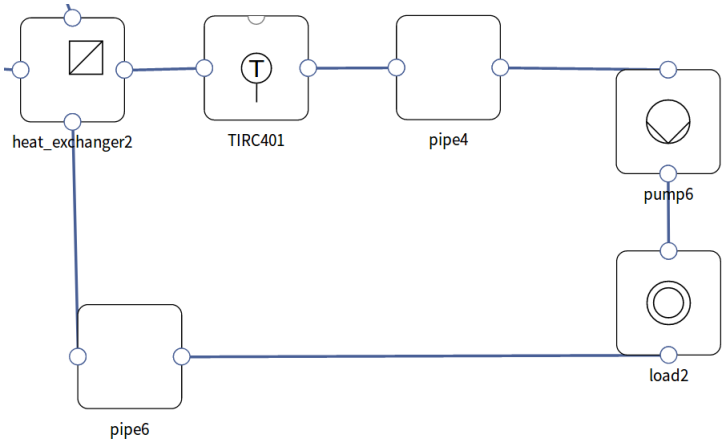


Abbildung 5.3.: Wasserkreislauf TWW

Der strukturelle Aufbau des Kreislaufs *TWW* entspricht, abgesehen von dem Temperatursensor *TIRC102*, auf den in diesem Kreislauf verzichtet wird, dem des Heizkreises. Die Elemente unterscheiden sich jedoch bezüglich ihrer Parametrierung. Hier wurden die Komponenten des Frischwasserzulaufs sowie der -zapfung ebenfalls durch eine Last vereinfacht dargestellt, da in diesem Kreislauf für die Regelung der iWÜST nur die sekundärseitige Vorlauftemperatur gemessen wird (siehe Abbildung 5.3).

Tabelle 5.1.: Hydraulische Komponenten des Modells

Typ	Anzahl	Komponente(n)
Pump_control_signal	4	MV101, MV301, MV302, P401
Pump	2	pump4, pump5
Heat_Exchanger	2	heat_exchanger1, heat_exchanger2
CHPPlant	1	generator
Load_From_Timeseries	2	load1, load2
Storage_2ConnectionPieces	2	storage1, storage2
Pipe	11	pipe1, pipe2, pipe3, pipe4, pipe5, pipe6, pipe7, pipe8, pipe9, pipe10, pipe11
T_Piece	6	t_piece1, t_piece2, t_piece3, t_piece4 t_piece5, t_piece6

5.2. Komponenten

Die Komponenten des Modells werden in diesem Abschnitt beschrieben. Auf die einzelnen Komponenten und deren Parameter wird genauer eingegangen, falls diese nicht den *default*-Werten der Bibliothek *component_library* entsprechen.

5.2.1. Hydraulische Komponenten

In Tabelle 5.1 sind die im Modell enthaltenen hydraulischen Komponenten aufgeführt. Die Verschaltung dieser Komponenten kann den Abbildungen 5.1, 5.2 und 5.3 entnommen werden.

Pumpen

Die Pumpen *pump5* und *pump6* besitzen einen festen Massenstrom. Dieser ist so gewählt, dass sich bei minimaler und maximaler Last eine realistische Vor- und Rücklauftemperatur aufgrund der Wärmeabnahme der Last einstellt. Die Pumpe *P401* soll so geregelt werden, dass die Vorlauftemperatur am Sensor *TIRC401* im Kreislauf TWW einem festgelegten Sollwert entspricht. Die übrigen Pumpen *MV101*, *MV301* und *MV302* stellen im drucklosen Modell die Ventile dar. Der maximal mögliche Massenstrom von *MV301* ist größer als der der anderen Pumpen, da diese Pumpe unter bestimmten Umständen den Massenstrom für die Speicherladung bereitstellen und den Wärmeübertrager der TWW versorgen können muss. *MV101* wird zur Regelung der Temperatur *TC101* verwendet, *MV301* stellt, je nach Szenario, den SOC des Speichers bzw. die Temperaturen *TIRC323*, *TIRC322* und *TIRC324* ein.

Tabelle 5.2.: Parameter der Pumpen im Modell

Komponente	Fördermenge	Regelparameter	Typ
pump5	0,3063 kg/s		Pump
pump6	0,35 kg/s		Pump
MV101	0..0,7 kg/s	TC101/TIRC103	Pump_control_signal
MV301	0..1,7 kg/s	SOC, TIRC322, TIRC323, TIRC324	Pump_control_signal
MV302	0..0,7 kg/s	TIRC321, TIRC302	Pump_control_signal
P401	0..0,7 kg/s	TIRC401	Pump_control_signal

Die Reglereinstellungen für Pumpe und Ventil sind bezüglich ihrer Auswirkungen auf die Aktoren ähnlich. Das Öffnen eines Ventils in der realen Anlage entspricht dem Erhöhen der Pumpenleistung in der Simulation, da beide Aktionen eine Erhöhung des Massenstroms zur Folge haben.

Die Pumpe *MV302* simuliert ein Mischventil. Bei maximaler Signalstärke der Steuerung fördert diese Pumpe den gleichen Massenstrom wie Pumpe *P401*. Somit wird das Wärmeträgermedium in den Speicher geleitet. Liegt die minimale Signalstärke an, so wird von der Pumpe nichts gefördert und das Medium wird direkt in den Rücklauf der TWW gespeist. Mit dem aktuellen Entwicklungsstand der Simulationsumgebung sind Massenströme mit dem Wert 0 kg/s aufgrund einer Nulldivision und dem dadurch bedingten Simulationsabbruch nicht möglich. Deswegen liegt dann ein Signal an, welches geringfügig über Null liegt. In Tabelle 5.2 sind die Parameter der einzelnen Pumpen enthalten.

Wärmeverbraucher

Im System sind die Lasten *load1* und *load2* vom Typ *Load_From_Timeseries* enthalten. Die Leistungsabnahme der Lasten wird durch eine Zeitreihe bestimmt, die Leistungswerte in Sekundenschritten repräsentiert. Die maximale Leistungsabnahme beider Lasten beträgt 115,5 kW und liegt damit 10 % über der Leistungsbegrenzung der realen Anlage (siehe Kapitel 6.6.2). Die maximale Lastabnahme für die Last *load2* wurde mittels der Norm *DIN END12831-3:2017* unter Verwendung des Profils *Krankenhaus* und dem Bedarf für eine *Sportstätte* ermittelt, da es keine Werte für Polizeigebäude gibt und die zeitliche Gebäudenutzung der Polizei einem Krankenhaus ähnelt. Der Bedarf einer *Sportstätte* wurde verwendet, da für die Simulation davon ausgegangen wurde, dass die Bediensteten nach dem Dienst die Dusche verwenden. In Anlehnung an die Prüfverfahren für Wärmepumpen in *DIN EN16147:2017* wurde eine Lastkurve für Spitzenlasten erstellt. Bei einem hohen Gleichzeitigkeitsgrad der Verwendung der einzelnen Zapfstellen tritt eine Spitzenlast von

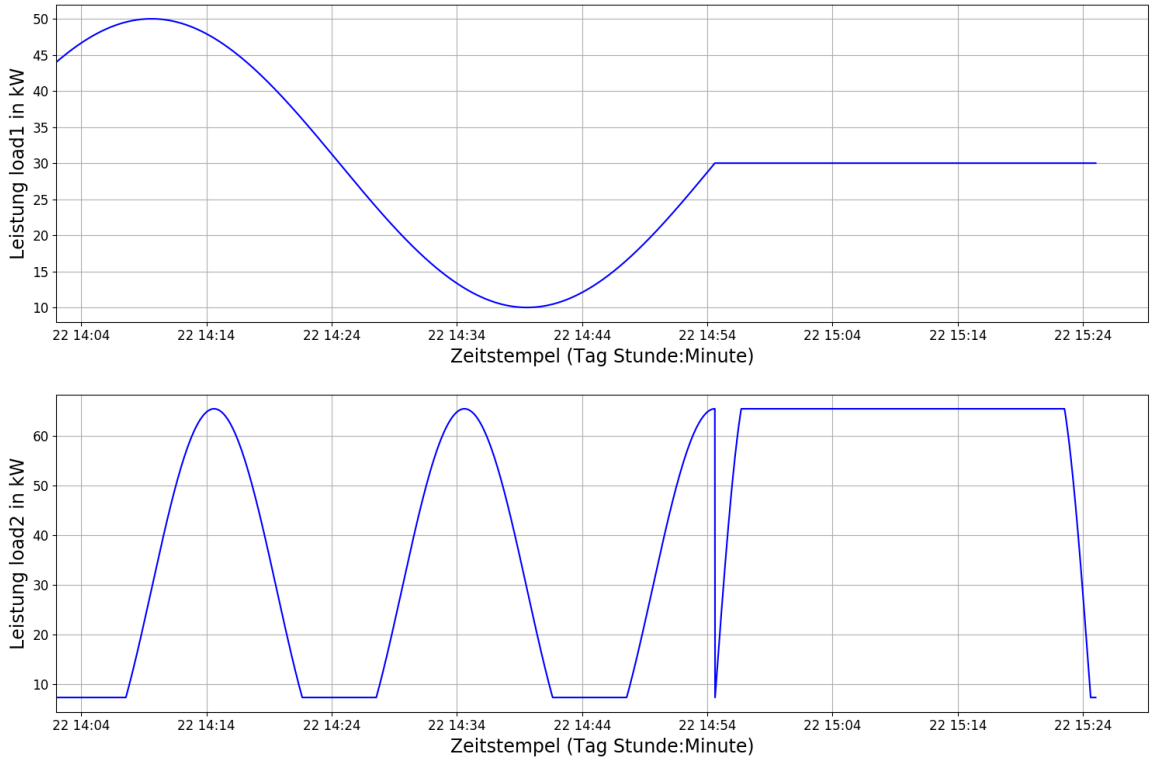


Abbildung 5.4.: oben: Simulierte Lastkurve *load1*; unten: Simulierte Lastkurve *load2*

65 kW auf. Für die Lastprofile wurden absichtlich steilere Profile als die in der Realität vorkommenden angenommen. Dadurch soll sichergestellt sein, dass die Regelung in der Lage ist, extreme Situationen bei der Lastabnahme auszuregeln. Außerdem wurde bei den Lastprofilen berücksichtigt, dass unter Verwendung des Simulationsmodells die Szenarien aus Kapitel 2.2.3 ausgelöst werden. Manche Szenarien werden durch Systemzustände ausgelöst, die durch die Lastkurven provoziert werden.

Die Lastkurven der Lasten *load1* und *load2* sind in Abbildung 5.4 gezeigt. Die Lastkurve von *load1* folgt einer Sinusschwingung mit anschließender konstanten Lastabnahme von 30 kW. Die Leistungsabnahme der Last *load2* verläuft zunächst sinusförmig, wobei der untere Bereich abgeschnitten ist. Die minimal abgenommene Leistung beträgt 7,3 kW. Nach der dritten Schwingung fällt die Leistung kurz ab, woraufhin sie anschließend sprunghaft auf den maximalen Wert von 65,5 kW ansteigt. Die Last ist anschließend für 29 Minuten konstant bei 65,5 kW und fällt zum Schluss steil auf die minimale Leistung ab.

Tabelle 5.3.: Parameter für die Berechnung der *kA*-Werte

Komponente	**max. Leistungsabnahme**	**max.** Δt	**min.** Δt
heat_exchanger1	50 kW	20 °C	3 °C
heat_exchanger2	65 kW	30 °C	1 °C

Wärmeübertrager

Die Wärmeübertrager *heat_exchanger1* und *heat_exchanger2* übertragen die Wärmeenergie des Kreislaufs *Wärmenetz* in die Kreiseläufe *Heizkreis* und *TWW*. Der Parameter *kA* der einzelnen Wärmeübertrager wurde mit Hilfe der Formel 2.1 wie folgt berechnet:

$$k \cdot A_1 = \frac{\dot{Q}}{\Delta T_m} = \frac{50000W}{8,96°C} = 5580\frac{W}{°C} \tag{5.1}$$

$$k \cdot A_2 = \frac{\dot{Q}}{\Delta T_m} = \frac{65000W}{8,52°C} = 7629\frac{W}{°C} \tag{5.2}$$

Der Wert für ΔT_m wurde nach Formel 2.2 berechnet. Die verwendeten Parameter für die Berechnungen können Tabelle 5.3 entnommen werden. Die Werte der Temperaturen für die Berechnung von ΔT_m konnten durch interne Erfahrungswerte an einer realen Anlage herangezogen werden.

Wärmeerzeuger

In dem Modell ist die Komponente *generator* vom Typ *CHPPlant* integriert. Der Erzeuger wird von einem PID-Regler geregelt. Dabei soll eine festgelegte Temperatur als Vorlauftemperatur des Wärmenetzes eingestellt werden, welche vom Temperatursensor *temp_sensor_help* ausgelesen wird. Die Temperatur am Sensor *temp_sensor_help* ist die Regelgröße des PID-Reglers, die zugeführte Wärmemenge des *generators* die Stellgröße. Der Wärmeerzeuger simuliert das Wärmenetz, welches in der Realität ebenfalls eine stabile Vorlauftemperatur besitzen soll. Der Wärmeerzeuger ist in der Simulation auf eine maximale Leistung von 250 kW begrenzt, wobei jedoch sichergestellt ist, dass bei maximaler Abnahme der Lasten (115,5 kW) die Erzeugung den Bedarf decken kann.

Wärmespeicher

Die reale iWÜST mit TWW enthält einen Wärmespeicher mit mittlerem Ladestutzen. Dessen Umsetzung in der Simulationsumgebung ist in dem Schema in Abbildung 5.5 dargestellt.Zwei

Speicher vom Typ *Storage_2ConnectionPieces* sind durch ein T-Stück gekoppelt. Die beiden Speicher *storage1* und *storage2* wurden mit jeweils drei Schichten parametriert und besitzen somit drei Konnektoren für Temperatursignale. Da der reale Speicher aber nur über fünf Temperatursensoren verfügt, werden vom *storage2* lediglich der untere und mittlere Sensor verwendet. Es kann außerdem angenommen werden, dass die nicht miteinbezogene obere Temperatur von *storage2* annähernd der des unteren Fühlers von *storage1* entspricht.

Das Gesamtvolumen der beiden gleich großen Speicher beträgt 1507l. Dies entspricht dem Volumen von üblichen Wärmespeichern in Wärmeübergabestationen und ist in der Größenordnung des Speichers der realen Anlage. Für die Simulation wurden die Parameter der Isolation mit Hilfe der Berechnung realistischer Temperaturverluste festgelegt, da die *default*-Werte der Komponente einen zu hohen Wärmeverlust erzeugten (siehe Abbildung A.10 im Anhang). Infolgedessen wurden die dafür verantwortlichen Kennzahlen *U_Wall* = 0,0001, *U_Top* = 0,0001 und *U_Bottom* = 0,0001 für beide Speicher angepasst.

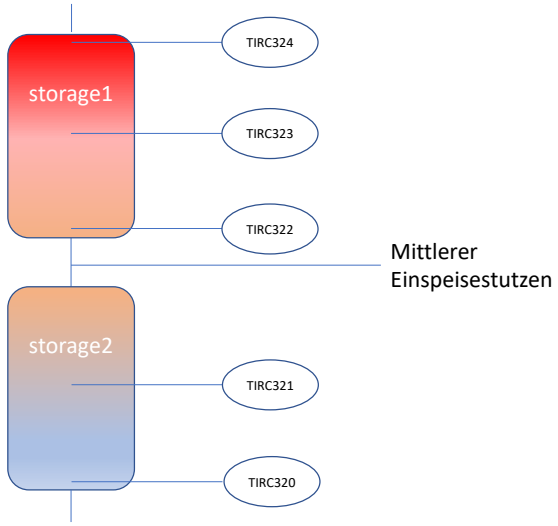


Abbildung 5.5.: Anordnung der Temperaturfühler in den Speichern

5.2.2. Steuerungstechnische Komponenten

Die im Modell enthaltenen steuerungstechnischen Komponenten sind in Tabelle 5.4 aufgeführt. Im System werden 13 Temperaturen erfasst. Die Temperaturen *TIRC320*, *TIRC321*, *TIRC322*, *TIRC323* und *TIRC324* werden intern in den Speichern *storage1* und *storage2* berechnet und an die Klemmen *plc_temp3* und *plc_temp4* geschickt. Die übrigen sieben Temperaturen, welche in der realen iWÜST erfasst werden, werden über Temperatursensoren vom Typ *Temperature_Sensor* eingelesen. Die Platzierung und Benennung der Temperatursensoren kann

Tabelle 5.4.: Steuerungstechnische Komponenten

Typ	Anzahl	Komponenten
plc_cpu	1	plc_cpu1
PLC_PT1000_Card	4	plc_temp1, plc_temp2, plc_temp3, plc_temp4
PLC_Analog_Output_Card	3	plc_ao1, plc_ao2, plc_ao3
PLC_End_Card	1	plc_end
Temperature_Sensor	8	TC101, TIR301, TIRC102, TIRC103, TIRC302, TIRC305, TIRC401, temp_sensor_help

den Abbildungen der Wasserkreisläufe aus Kapitel 5.1 entnommen werden. Ein zusätzlicher Sensor (*temp_sensor_help*), wird für die Regelung der Komponente *generator* benötigt. Die Temperaturen werden als Werte mit der Einheit Kelvin an die Klemmen übergeben, wohingegen die Klemmen der realen SPS die Temperaturen einlesen und an die SPS im Format $^{\circ}C \times 10$ weitergeben. Dies musste bei der Verwendung des Anwenderprogramms der SPS der Hardwaretestumgebung berücksichtigt werden.

5.3. Einstellung der Regelparameter

Die Regelung der verschiedenen Aktoren des Modells wurde mit Hilfe von PID-Reglern der Klasse *PID* aus der Simulationsumgebung umgesetzt. Die Regelgrößen *TIRC401*, *TC101*, *SOC* und *temp_sensor_help* werden von einem jeweiligen Regler eingestellt. Für eine ausreichend genaue Reglereinstellung genügte die Verwendung des Proportional- und Integralglieds der Regler.

Allgemeines Vorgehen bei der Parametrierung

Die Regelparameter wurden für die einzelnen Regler nacheinander nach der *Empirischen Methode zur Ermittlung der Regelparameter* eingestellt. Dabei wird zunächst der P-Anteil (*KP*) so lange um den Faktor zehn erhöht, bis sich ein zügiges Einschwingung (2-3 Überschwinger) der zu regelnden Größe einstellt. Anschließend kann der Verstärkungsfaktor *KP* in kleineren Schritten angepasst werden, falls die Amplitude der Schwingung zu groß ist. Zur Minimierung der Regelabweichung wird anschließend der Wert *TN* angepasst, bis das gewünschte Regelverhalten erreicht ist.[24]
Die sonstigen Parameter für die Reglereinstellung wurden nicht verändert und entsprechen somit den *default*-Werten der Klasse *PID*. Die Einstellungen wurden am Simulationsmodell durchgeführt, das in den Kapiteln 5.1 und 5.2 beschrieben wird. Die einzelnen Regelungen wurden zunächst bei konstanten Leistungswerten der Lasten eingestellt und anschließend

Tabelle 5.5.: Regler des Simulationsmodell

Regler	**Regelgröße**	**Stellglied**
pid_generator	Temperatur von *temp_sensor_help*	$\dot{Q}$ des Erzeugers *generator*
pid_MV101	Temperatur von *TC101*	$\dot{m}$ der Pumpe *MV101*
pid_MV301	SOC des Speichers / Temperaturen *TIRC322*, *TIRC323*, *TIRC324*	$\dot{m}$ der Pumpe *MV301*
pid_P401	Temperatur von *TIRC401*	$\dot{m}$ der Pumpe *P401*

durch Verwendung der Lastprofile aus Kapitel 5.2.1 bzgl. der Reaktion auf sprunghafte Veränderungen validiert.
Im Folgenden werden Simulationen, bei denen die Lastprofile und die in den vorherigen Kapiteln beschriebenen Parameter verwendet wurden, als *Testsimulation* bezeichnet. Zu jeder Regelung wurden die simulierten Werte der Regelgröße und der Stellgröße grafisch ausgewertet. Bezüglich der Qualität des Regelverhaltens ist zu berücksichtigen, dass für die Untersuchungen, die mit Hilfe des Modells durchgeführt werden, kein Regelverhalten "aus dem Lehrbuch" benötigt wird. Leichte Regelabweichungen und ein geringes Einschwingen sind demnach akzeptierbar.
In Tabelle 5.5 sind die Regler, die im Simulationsmodell verwendet werden, mit der jeweiligen Regelgröße und dem Stellglied aufgeführt. Die Einstellung der Parameter wird in den folgenden Abschnitten beschrieben.

Regelung des Wärmeerzeugers *generator*

Für die Einstellung der Parameter des Reglers *pid_generator* wurde den übrigen Aktoren konstante Werte zugewiesen, wobei die TWW nicht durchströmt wurde. Zunächst wurde der *KP*-Wert auf $5 \cdot 10^{-7}$ gesetzt. Der *TN*-Wert wurde mit *np.inf* angegeben (Parametrierung I). Das Kürzel *np.inf* repräsentiert in der Programmiersprache Python den Wert unendlich. Die Regelgröße *temp_sensor_help* (*ACTUAL*) soll mit Hilfe des Reglers auf den Sollwert 363,15K (*SET_POINT*) geregelt werden. Dabei wurde in den Simulationsergebnissen deutlich, dass neben einem sehr langsamen Einschwingverhalten ein anfänglicher Ausschlag der Temperatur vorhanden ist (siehe Abbildung 5.6).
Daraufhin wurde der *KP*-Wert um den Faktor zehn erhöht und der Initalwert des *Analog_Drive_Controller*, welcher den Ausgangswert des PID-Reglers in den Grenzen 0,004..0,02 hält, von 0,012 auf 0,004 gestellt. In der folgenden Simulation war der anfängliche Ausschlag der Temperatur nicht mehr vorhanden, jedoch war das Einschwingverhalten immer noch zu langsam (Abbildung A.11 im Anhang). Das erwünschte Verhalten wurde im nächsten Schritt erreicht, als der *KP*-Wert auf $5 \cdot 10^{-5}$ eingestellt wurde. Zum Erreichen des Sollwertes wurde die Nachstellzeit *TN* auf $1 \cdot 10^{7}$ gesetzt (Parametrierung II). Der Temperaturverlauf

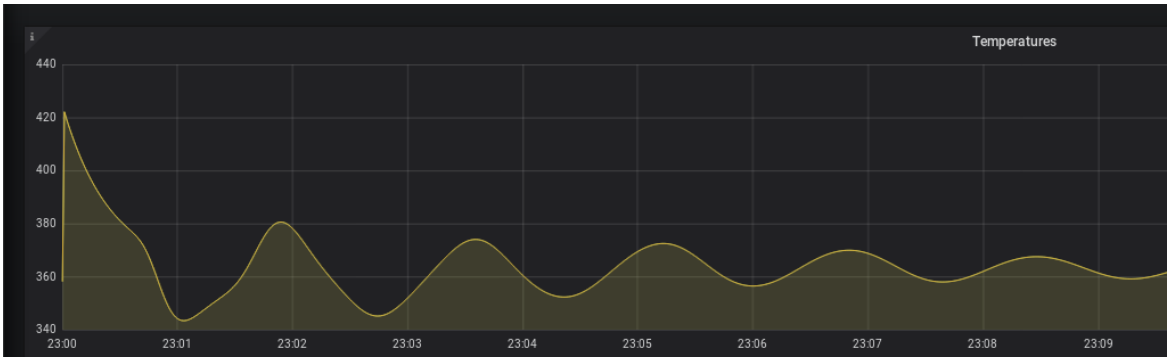


Abbildung 5.6.: Temperaturverlauf des Sensors temp_sensor_help (Parametrierung I)

des Sensors *temp_sensor_help* in Abbildung 5.7 zeigt zunächst ein gutes Einschwingverhalten. Anschließend sinkt die Temperatur kurzzeitig auf 358 K, was auf eine sehr geringe Initaltemperatur in einer der Systemkomponononenten zurückzuführen ist. Anschließend wird der Sollwert mit geringem, akzeptierbaren Über- und Unterschwingen von 0,5 K eingestellt. Bei paralleler Durchstörmung der beiden Wärmeübergabestationen *heat_exchanger1* und

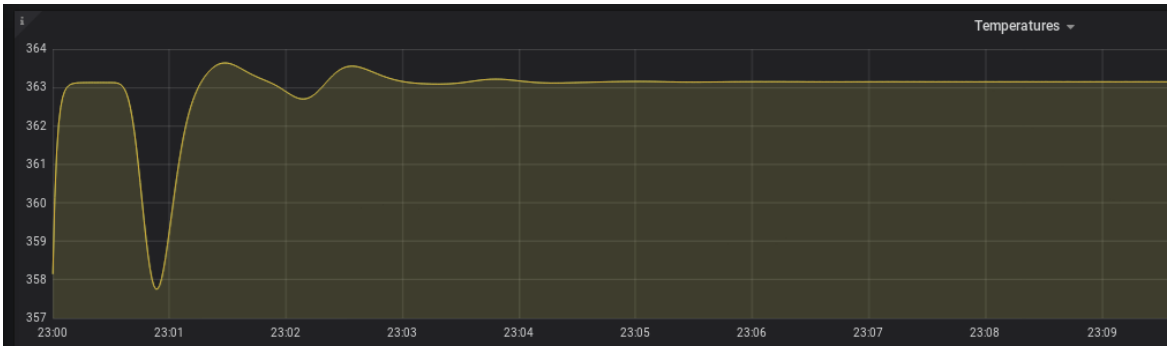


Abbildung 5.7.: Temperaturverlauf des Sensors temp_sensor_help (Parametrierung II)

heat_exchanger2 regelt der PID-Regler ebenfalls zufriedenstellend.
Abbildung 5.8 zeigt den Temperaturverlauf der Regelgröße *temp_sensor_help* bei Verwendung aller Regelungen der *Testsimulation*, wobei der Sollwert des SOC konstant bei 80 % lag. Sollwerte sind in den folgenden Grafiken als orange, gestrichelte Linie eingezeichnet. Der Sollwert der Temperatur beträgt 343 K. Dieser Wert wird nach einem kurzen Einschwingverhalten aufgrund der unterschiedlichen Starttemperaturen der Systemkomponenten ausreichend genau durch den Regler eingestellt. Das vom PID-Regler *pid_generator* erzeugte Steuersignal in Milliampere ist in Abbildung 5.9 dargestellt. In der Grafik ist, abgesehen von dem kurzen Einbruch des Steuersignals aufgrund von unterschiedlichen Initalwerten im Bereich des Zeitpunkts 14:05, die Überlagerung der beiden Lastkurven und ab dem Zeitpunkt 15:04 das Be- und Entladen des Speichers zu erkennen.

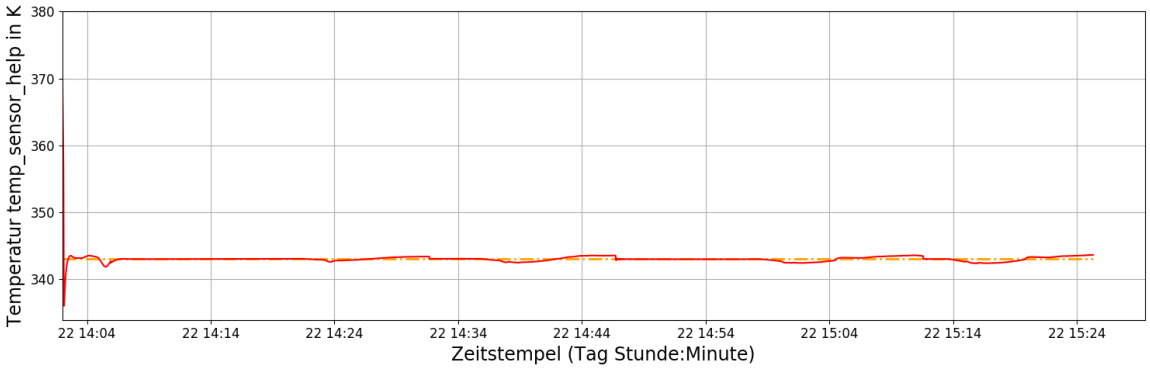


Abbildung 5.8.: Temperaturverlauf *temp_sensor_help* (*Testsimulation*)

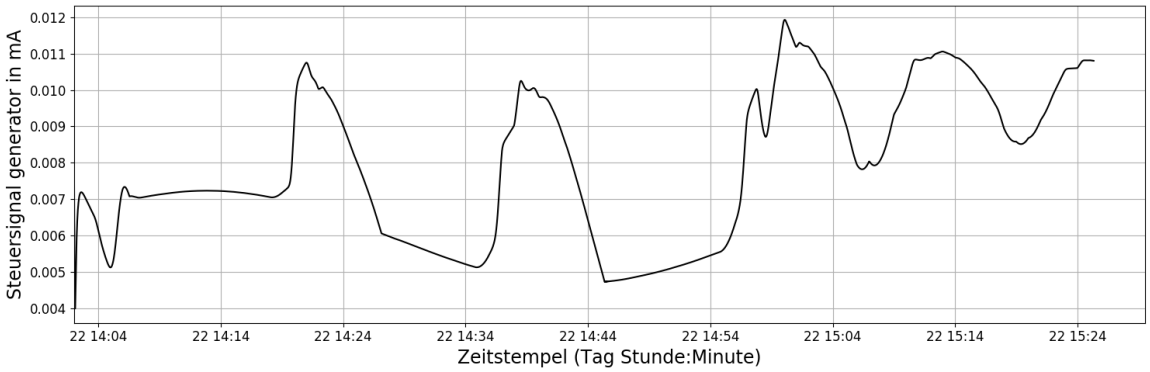


Abbildung 5.9.: Steuersignal des Reglers *pid_generator*

Regelung der Pumpe *MV101*

Für die Parametrierung der Regelung *pid_MV101* wurde kein Volumenstrom im Kreislauf TWW und eine konstante Last für *load1* vorgegeben. Der Wärmeerzeuger wurde mit aktiver Regelung *pid_generator* verwendet, sodass sich eine realtiv konstante Netztemperatur ergibt. Trotzdem brach die Simulation auffällig häufig nach einigen Simulationsschritten ab. Nach Anpassung des konstanten Pumpenparameters *mass_flow* der Pumpe *pump5* im Kreislauf Heizkreis konnte die Simulation komplett berechnet werden. Bei der Betrachtung der sich ergebenen Kennlinien der Rohrleitung *pipe1* (Abbildung A.12 im Anhang) sowie des Temperatursensors *TC101* waren starke, symmetrisch widerkehrende “Zacken” in den Temperaturen und Massenströmen zu sehen. In Abbildung 5.10 ist der Temperaturverlauf des Sensors *TC101* dargestellt. Nachdem ausgeschlossen werde konnte, dass die Regelung *pid_MV101* für dieses Verhalten verantwortlich ist, wurde der Code des Wärmeübertragermodells (Methode *temperature* der Komponente *heat_exchanger*) genauer untersucht. Dabei fiel auf, dass bei einem Massenstrom unter $0{,}2\,\mathrm{kg/s}$ die Methode der Temperaturberechnung

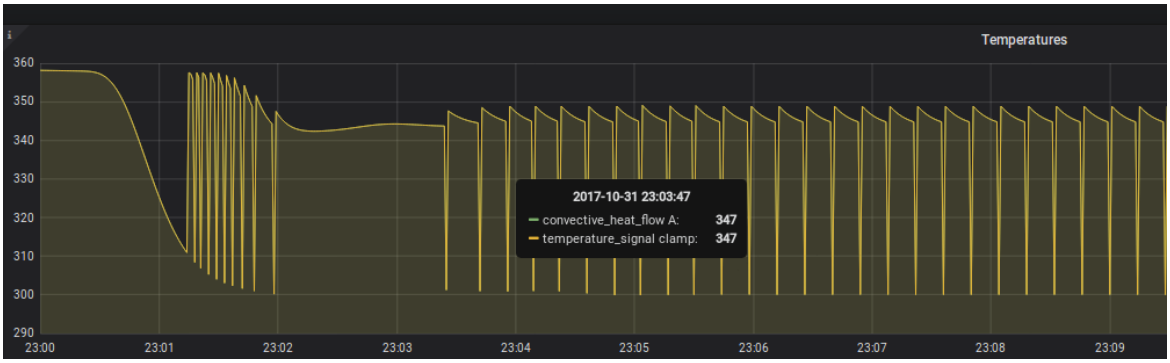


Abbildung 5.10.: Temperaturverlauf des Sensors *TC101*

als Ausgangstemperatur (sekundär) die Eingangstemperatur (primär) des Wasserkreislaufs wählt. Steigt der Wert des Massestroms über $0{,}2\,\mathrm{kg/s}$, so wird die Temperatur nach dem physikalischen Modell berechnet. Bei im Bezug auf die Größe des Wärmeübertragers sehr geringen Massenströmen führt die Berechnung der Ausgangstemperatur zu einem mathematischen Fehler, weshalb die Programmierung wie oben beschrieben umgesetzt wurde.[23]

In der Simulation schwankte der Wert des Massenstroms um den Wert $0{,}2\,\mathrm{kg/s}$, woraufhin die Temperatur zwischen berechneter und durchgereichter pendelt, wie in Abbildung 5.10 zu sehen ist. Um diesen Effekt zu vermeiden, wurde der Grenzwert in der Methode *temperature* der Komponente *heat_exchanger* in diesem Beispiel von $0{,}2\,\mathrm{kg/s}$ auf $0{,}001\,\mathrm{kg/s}$ reduziert. Nach Abschluss dieser Arbeit soll ein dynamisch berechneter Grenzwert in die Methode implementiert werden. Die Regelparameter konnten anschließend mit dem neuen Grenzwert nach der empirischen Methode bestimmt werden. Das gewünschte Regelverhalten wurde bei Verwendung der Parameter $Kp = 5 \cdot 10^{-5}$ und $TN = 1 \cdot 10^{7}$ erreicht.

Die von der Simulationsumgebung berechneten Verläufe der Regelgröße *TC101* sowie der Stellgröße *MV101* unter Verwendung der *Testsimulation* mit einem Ladezustand des Speichers von *SOC_soll*= $80\,\%$ sind in Abbildung 5.11 und 5.12 zu sehen. Der Sollwert *TC101_soll* = $323\,\mathrm{K}$ ($50\,^{\circ}\mathrm{C}$) wird nach dem Starten der Simulation in kurzer Zeit erreicht und über den betrachteten Zeitraum sehr genau von der Regelung gehalten. Die Sinuswelle der Lastkurve mit anschließendem konstanten Verlauf ist im Profil des Massenstroms der Pumpe deutlich erkennbar. Bei hoher Last fördert die Pumpe einen hohen Massenstrom, bei niedriger Last einen geringen Massenstrom.

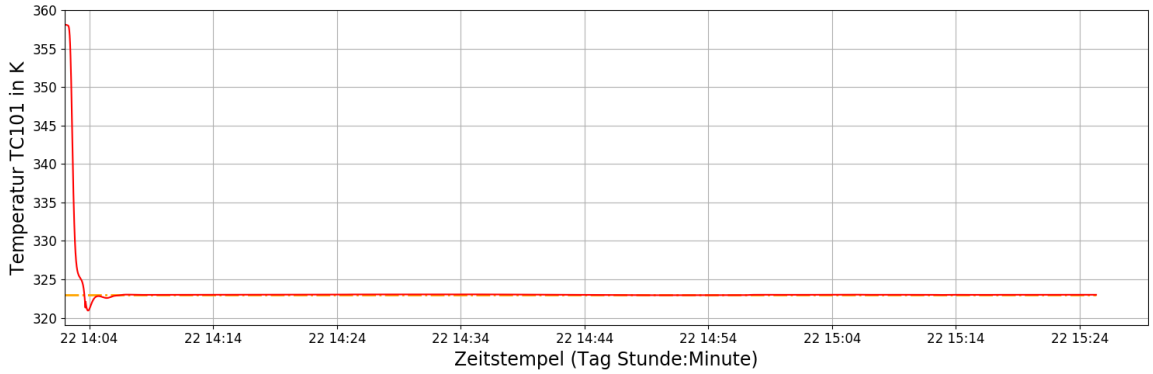


Abbildung 5.11.: Temperaturverlauf des Sensors *TC101*

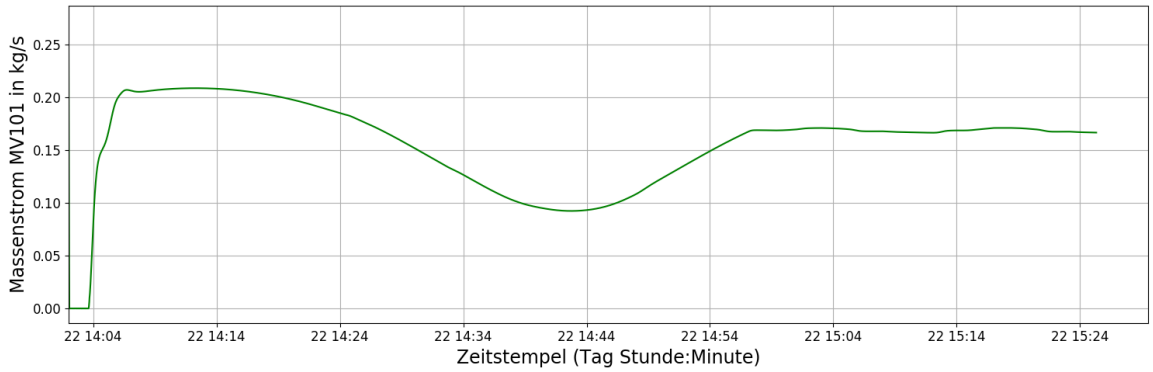


Abbildung 5.12.: Massenstrom der Pumpe *MV101*

Regelung der Pumpe *P401*

Der maximale Massenstrom der Pumpe *P401* beträgt $0{,}7\,\mathrm{kg/s}$ und ist somit genauso groß wie der maximale Massenstrom der Pumpe *MV101*. Die Pumpe *P401* soll die Temperatur *TIRC401* auf $333\,\mathrm{K}$ ($60\,^{\circ}\mathrm{C}$) regeln. Die Verwendung der Regelparameter von Pumpe *MV101* ($Kp = 5 \cdot 10^{-5}$, $TN = 1 \cdot 10^{7}$) führte zu einem gewünschten Regelverhalten, weshalb diese Parameter auch für den PID-Regler *pid_P401* übernommen wurden. Das gewünschte Regelverhalten ist anhand des Temperaturverlaufs des Sensors *TIRC401* und des Massenstromverlaufs der Pumpe *P401* bei Verwendung der *Testsimulation* in den Abbildungen 5.13 und 5.14 dargestellt. Im grafischen Verlauf des von der Pumpe geförderten Massenstroms ist das Lastprofil der Last *load2* erkennbar. Anhand des Temperaturverlaufs zeigt sich, in welchen Situationen größere Abweichungen vom Sollwert vorhanden sind. Zum Zeitpunkt 14:56 liegt der Istwert um $3\,\mathrm{K}$ über dem Sollwert. Dieser Verlauf wird durch den starken Anstieg nach vorherigem Abfallen der Last *load2* verursacht und entspricht so dem erwarteten Verhalten des Modells.

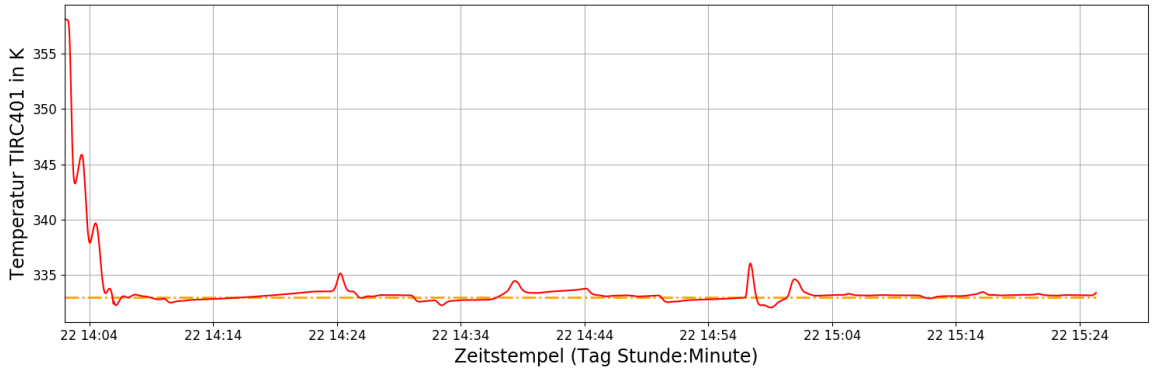


Abbildung 5.13.: Temperaturverlauf des Sensors *TIRC401*

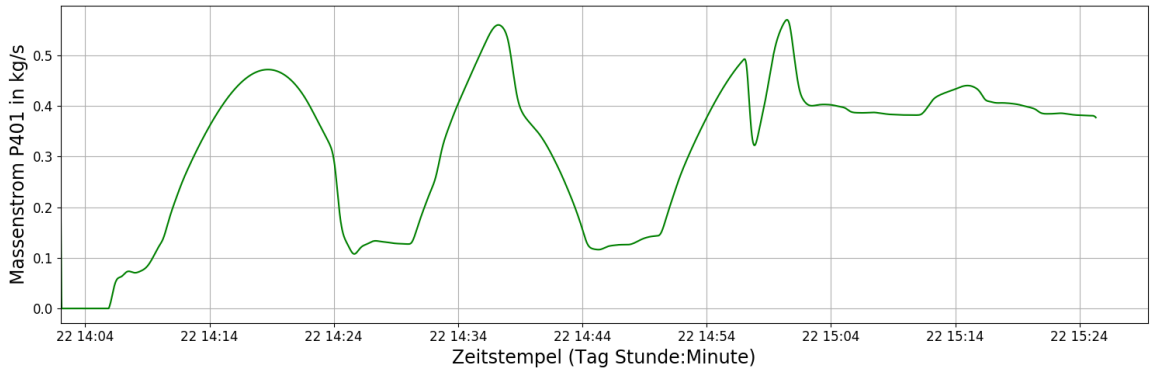


Abbildung 5.14.: Massenstrom der Pumpe *P401*

Regelung der Pumpe *MV301*

Die Pumpe *MV301* regelt, abhängig vom Szenario, den SOC oder die Temperaturen im Speicher, wobei in diesem Abschnitt lediglich auf die Regelung des SOC eingegangen wird. Die Regelung bzgl. der Temperaturen wird in Kapitel 6.6.2 beschrieben. Der SOC wird vom Regelalgorithmus wie folgt berechnet:

$$SOC = \frac{T_{med} - T\_RL_{min}}{T\_Netz_{mittel} - T\_RL_{min}} \cdot 100 \tag{5.3}$$

Die mittlere Speichertemperatur T_{med} ist das arithmetrische Mittel der fünf Speichertemperaturen. T_RL_{min} ist die minimal mögliche Rücklauftemperatur der Wärmeübergabestation (298K), T_Netz_{mittel} die gemittelte Temperatur des Vorlaufs der letzten 60 Sekunden. Die gemittelte Temperatur wird verwendet, da kurzzeitige Temperaturschwankungen im Netz ansonsten einen sich stark verändernden SOC bedingen würden. Dadurch würde die Regelung der Pumpe MV301 nicht wie gewünscht erfolgen und könnte im schlimmsten Fall

eine Dauerschwingung erzeugen. Die Regelung *pid_MV301* der Pumpe *MV301* soll träger als die Regelung *pid_P401* sein, damit zwischen den Regelungen kein Resonanzverhalten aufkommen kann und ein Aufschwingen der Stellgrößen vermieden wird. Der Wärmespeicher ist außerdem im Gegensatz zur Temperatur *TIRC401* eine trägere Regelstrecke, weshalb eine trägere Regelung von Vorteil ist.

Die Trägheit einer Regelung wird primär durch den *KP*-Wert bestimmt. Der *KP*-Wert der Regelung *pid_MV301* wurde zunächst um den Faktor zehn kleiner als der Wert der Regelung *pid_P401* = $5 \cdot 10^{-5}$ eingestellt. Die Regelung war dadurch jedoch zu träge, sodass der *KP*-Wert mit $2{,}5 \cdot 10^{-5}$ halb so groß wie der Wert der Regelung *pid_P401* gewählt wurde. Durch diese Parametrierung wurde das gewünschte Regelverhalten erreicht. Abbildung 5.15 zeigt den Verlauf des SOC. Der Sollwert des SOC liegt dabei, im Gegensatz zu den vorherigen beiden Abschnitten, bis zum Zeitpunkt 14:42 bei 70 %. Ab diesem Zeitpunkt wird der Sollwert auf 80 % gesetzt. Die mittlere Einspeisung des Speichers wurde für die Übersichtlichkeit der Simulationsergebnisse in der Steuerung deaktiviert. Es ist zu erkennen, dass der Sollwert

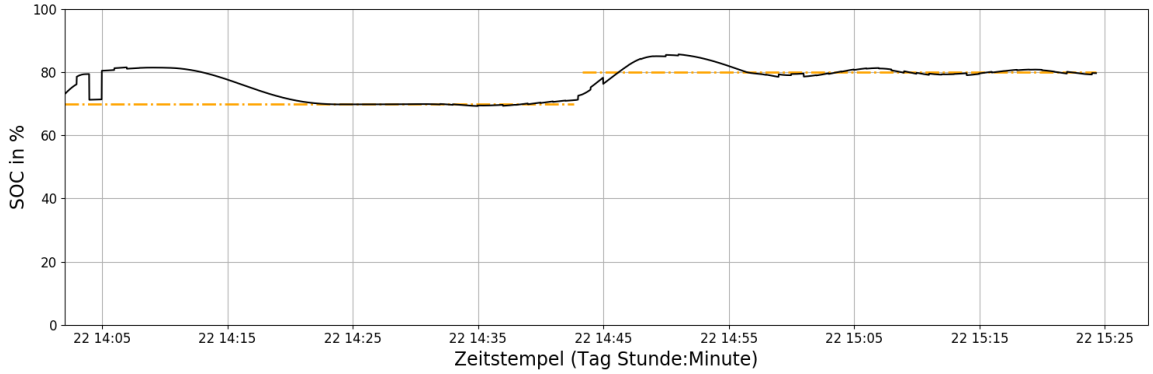


Abbildung 5.15.: SOC des gesamten Speichers

nach anfänglichem Überschwingen erreicht wird. Das kurze Abfallen des SOC im Zeitraum 14:04 - 14:05 ist auf eine Spitze in der Netztemperatur *temp_sensor_help* zurückzuführen (siehe Abbildung A.14 im Anhang). Durch das Bilden des Mittelwertes dieser Temperatur für die Berechnung des SOC (Formel 5.3) wird der Istwert für 60 Sekunden in die Bildung des Mittelwerts einbezogen. Würde man den SOC mit Momentanwerten der Netztemperatur berechnen, so läge der Wert für kurze Zeit bei 60 %. Die Veränderung des Sollwertes zum Zeitpunkt 14:42 wird von der Regelung zufriedenstellend ausgeregelt. Das leichte Über- und Unterschwingen des SOC kann dabei akzeptiert werden. Die Trägheit der Regelung sowie der Regelstrecke ist in der langsamen Reaktion auf den neuen Sollwert zu erkennen. Dies hat in diesem Systems keine negativen Auswirkungen und kann hingenommen werden. Das Verhalten der Pumpe *MV301* ist in Abbildung 5.16 dargestellt. Bis zum Zeitpunkt 14:24 wird der Speicher entladen. Der Sollwert ist ab diesem Zeitpunkt erreicht. Anschließend

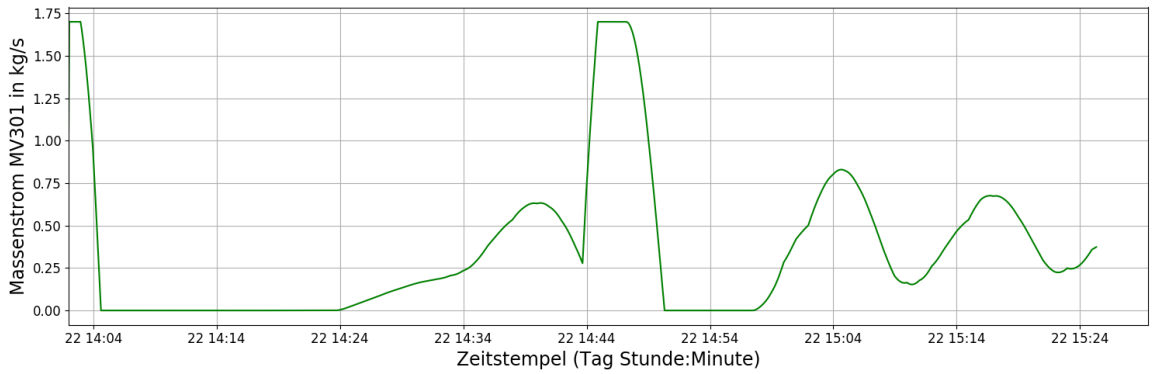


Abbildung 5.16.: Massenstrom der Pumpe *MV301*

folgt der zeitliche Verlauf des Massenstroms der Pumpe *MV301* der Lastkurve der Last *load2* aus Abbildung 5.4. Die Änderung des Sollwertes zum Zeitpunkt 14:42 ist durch den steilen Anstieg der Pumpenleistung der Pumpe *MV301* zu erkennen. Zum Zeitpunkt 14:46 ist der Sollwert erreicht und der Speicher besitzt den gewünschten Füllstand. Im weiteren Zeitverlauf ist das Einschwingen auf den Sollwert zu erkennen. Das Be- und Entladen des Speichers kann anhand Abbildung 5.17 betrachtet werden. Bei negativem Massenstrom im Speicher wird dieser entladen, bei positivem beladen. Die Kennlinien der weiteren Stell- und Regelgrößen sind in den Abbildungen A.15 - A.18 im Anhang abgebildet.

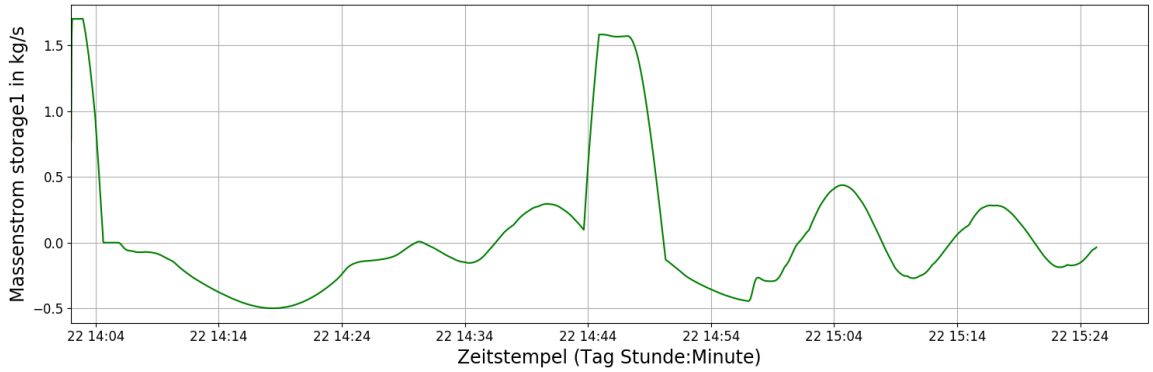


Abbildung 5.17.: Massenstrom im Speicher

Regelung der Pumpe *MV302*

Die Pumpe *MV302* simuliert das Drei-Wege-Ventil aus der realen Anlage. Dieses Ventil soll in der Realität, abhängig von der Rücklauftemperatur *TIRC302* und der mittleren Speichertemperatur *TIRC321*, den Massenstrom des Rücklaufs mittig in den Speicher einladen oder

am Speicher vorbeiführen. Ist die Rücklauftemperatur höher als die Temperatur am mittleren Einlaufstutzen (*TIRC302* > *TIRC321*), so soll der Massenstrom des Rücklaufs den Speicher laden (Abbildung 5.18 links). Bei geringerer Rücklauftemperatur (*TIRC302* < *TIRC321*) soll das Wärmeträgermedium nicht über den Speicher fließen (Abbildung 5.18 rechts).
Der Temperatursensor *TIRC321* wurde anstatt des Sensoren *TIRC322* verwendet, da dieser

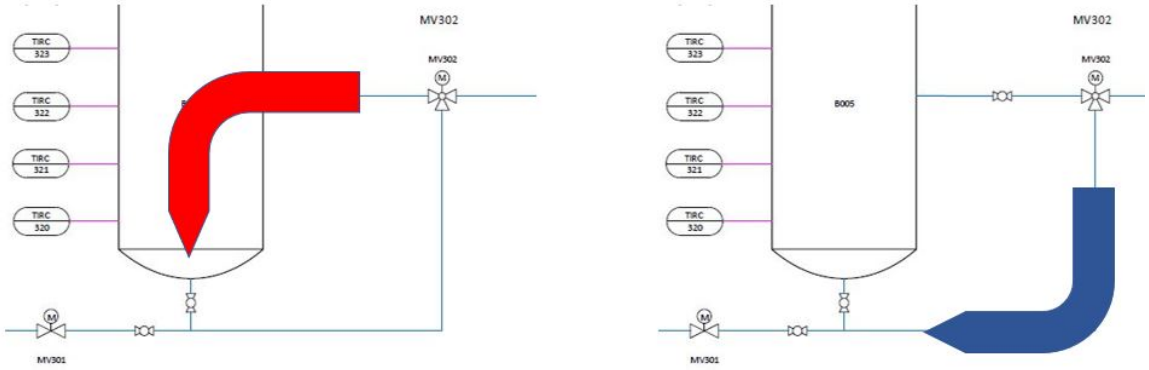


Abbildung 5.18.: links: Massenstrom bei geöffnetem Drei-Wege-Ventil *MV302*
rechts: Massenstrom bei geschlossenem Drei-Wege-Ventil *MV302*

nur die untere Temperatur im Speicher *storage1* misst (vgl. Abbildung 5.5) und so keine Veränderung der Temperatur erfassen kann. Im Falle des Beladens des Speichers ist ein Massenstrom nur durch den Speicher *storage2* vorhanden. Deswegen ändern sich nur dessen Temperaturen.
In der Simulation ist dieses Verhalten durch die Pumpe *MV302* im Strang zum mittleren Einspeisestutzen umgesetzt. Fördert diese Pumpe nicht, so wird der gesamte Massenstrom aus dem Rücklauf am Speicher vorbeigeführt. Ist der Rücklauf wärmer als die Temperatur am Temperatursensor *TIRC321*, so soll die Pumpe den gleichen Massenstrom wie die Pumpe *P401* fördern. Dadurch wird der gesamte Massenstrom aus dem Rücklauf in den Speicher geleitet. Dies wurde mit Hilfe des Regelalgorithmus im Anwenderprogramm implementiert. Das gewünschte Verhalten kann anhand der aufgezeichneten Verläufe in der Abbildung 5.19 betrachtet werden. Sobald die Temperatur des Sensors TIRC302 über die Temperatur des Sensors TIRC321 steigt, erhält die Pumpe *MV302* das gleiche Steuersignal wie die Pumpe *P401*. Die Pumpen sind bzgl. der maximalen und minimalen Leistung gleich parametriert, weshalb sie bei gleichem Steuersignal die gleiche simulierte Menge an Wärmeträgermedium fördern. Die Form des Verlaufs der Pumpe *MV301* wird dadurch von der Lastabnahme der Last *load1* bestimmt.

Fazit

Die Regelgrößen folgen dem Sollwert bei Verwendung der *Testsimulation* bei extremen Lastsprüngen mit leichtem Über- bzw. Unterschwingen. Dabei beträgt der größte Unterschied zwischen Ist- und Sollwert der Temperatur am Temperatursensor *TIRC401* 3 K. Die maximale Abweichung des SOC liegt nach dem anfänglichen Einschwingen des Systems bei rund 7 %. Dieser temporäre Unterschied hätte in der Realität keine signifikanten negativen Auswirkungen auf das System. Die Abweichungen der Temperaturen und des SOC von den Sollwerten zu Beginn der Simulation sind durch die unterschiedlichen Initaltemperaturen in den verschiedenen Komponenten begründet und können deswegen vernachlässigt werden. Die Regelparameter sind somit zufriedenstellend parametriert.

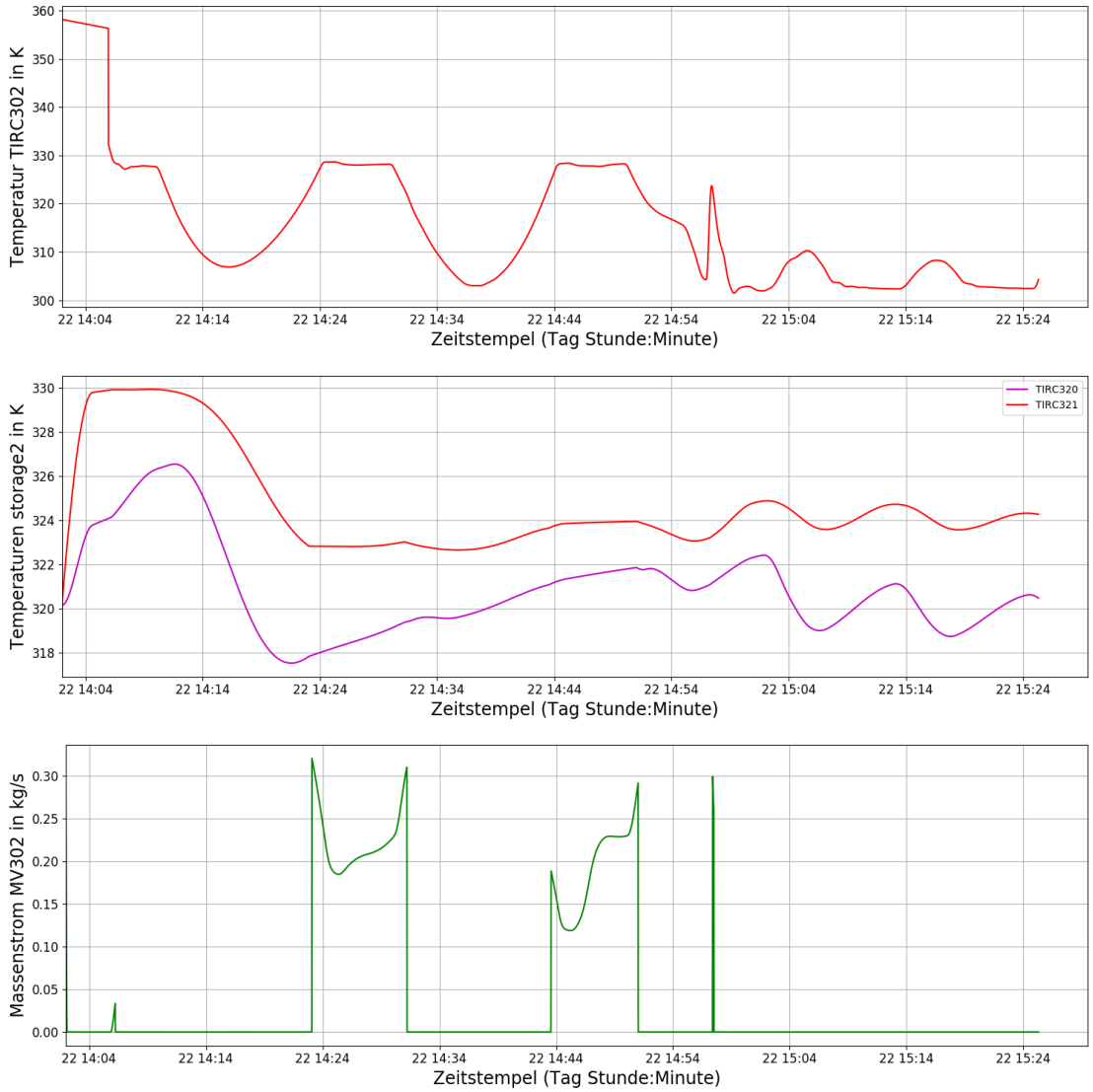


Abbildung 5.19.: Zeitlicher Verlauf des Sensors *TIRC302* (oben), der Sensoren *TIRC320* und *TIRC321* im Speicher *storage2* (mitte) und des Massenstroms der Pumpe *MV302* (unten)

6. Regelalgorithmus der iWÜST

Das folgende Kapitel beschreibt die Umsetzung eines Regelalgorithmus der iWÜST zur Versorgung eines Heizkreises sowie zur TWW. Dabei wird zunächst auf die Konfiguration der Systemkomponenten der SPS eingegangen. Anschließend folgt eine Beschreibung der Regelung sowie deren Realisierung. Die Validierung des Regelalgorithmus wird in Kapitel 6.6 durchgeführt. Zum Schluss wird auf die Ergebnisse und Probleme hinsichtlich der Modellannahme und des Regelalgorithmus eingegangen.

6.1. Konfiguration der SPS

In *CoDeSys 2.3* wurden die einzelnen Klemmen mit den angeschlossenen Aktoren, Sensoren und Schaltern gemäß Kapitel 4 in den *Steuerungskonfigurationen* definiert. Dabei wurde darauf geachtet, dass die Klemmanschlüsse gemäß der Namenskonvention aus Kapitel 3.4 benannt sind. Bei der folgenden Beschreibung des Regelalgorithmus wird für einen besseren Lesefluss bei den Temperatursensoren auf den Suffix *_clamp* bzw. den Präfix *storage_* verzichtet. Die genauen Bezeichnungen und Anschlüsse können der Export-Datei im Anhang A.4 entnommen werden.
Das Steuerungsprogramm der iWÜST enthält vier Tasks, welche in den *Taskkonfigurationen* beschrieben sind. Diese rufen in einer bestimmten Reihenfolge Anwenderprogramme auf. Die Reihenfolge ist von der Parametrierung (bspw. Aufrufintervall, Priorität) der Tasks abhängig (Tabelle 6.1 auf Seite 79). Die Priorität bestimmt, welcher Task bei gleichem Aufrufintervall zuerst aufgerufen wird. Das Aufrufintervall gibt an, in welchem zeitlichen Intervall die Programme aufgerufen werden. Das Aufrufintervall aller Tasks beträgt 200 ms.
Der Programmablaufplan des gesamten Regelalgorithmus ist in Abbildung 6.1 dargestellt. Der PID-Regler *pid_generator* ist in *CoDeSys 2.3* zwar angelegt, jedoch wird er in der Hardwaretestumgebung nicht verwendet. Der Regler wird ausschließlich in der Simulationsumgebung zur Regelung der Netztemperatur benötigt. Damit der PID-Regler nicht nachträglich der Export-Datei hinzugefügt werden muss, wird dieser schon in die Regelung der Hardwaretestumgebung mit der Flag *FLAG_JARVIS* implementiert (siehe Kapitel 6.2). In der Simulationsumgebung wird das Ausgangssignal des Reglers *pid_generator* an die Komponente *generator* gesendet.

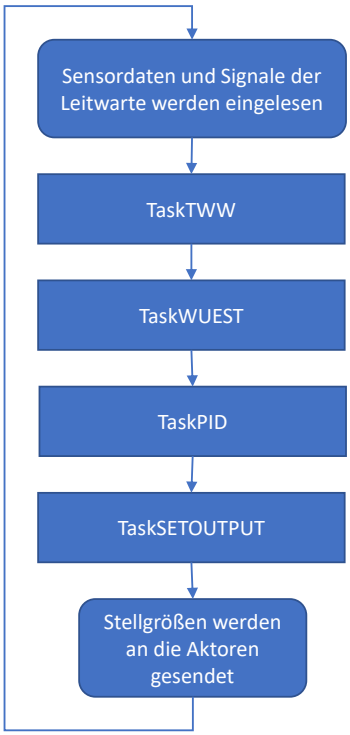


Abbildung 6.1.: Programmablaufplan der Steuerung

In den *Zielsystemeinstellungen* ist die verwendete WAGO-SPS *WAGO_750-880_-_750-880-040-000_&_750-885* konfiguriert. Die IP-Adresse der SPS sowie das *transport protocol* “tcp” sind in Bereich der *Kommunikationsparameter* definiert.

6.2. Vorkehrungen zur Verwendbarkeit des Regelalgorithmus im Hard- und Softwaremodell

Die Einschränkungen hinsichtlich der Verwendung von Export-Dateien und dem darin enthaltenen ST-Code in der Simulationsumgebung *Jarvis* wurden in Kapitel 3 beschrieben. Die Anwenderprogramme des Regelalgorithmus wurden so umgesetzt, dass diese durch geringfügiges Anpassen der Export-Datei in der Simulationsumgebung lauffähig sind. Dafür müssen die Variablen aus Tabelle 6.2 je nach Entwicklungsumgebung angepasst werden.

Durch die Verwendung eines Skalierungsfaktors und eines Offsets können die unterschiedlichen Datentypen der Temperaturen (K und $°C \times 10$) der Hardwaretestumgebung und der Simulationsumgebung im Regelalgorithmus verwendet werden. Die Unterschiede in der Verarbeitung der Signale durch die Ausgangsklemmen wurden durch die Verwendung von

Tabelle 6.1.: Taskkonfigurationen der SPS

Task	**Priorität**	**Programm(e)**	**Beschreibung**
TaskTWW	1	PLC_PRG_TWW()	Zustandsbeschreibung und Sollwertdefinition der iWÜST mit TWW
TaskWUEST	2	PLC_PRG_WUEST()	Zustandsbeschreibung und Sollwertdefinition der iWÜST ohne TWW
TaskPID	3	PID_generator() PID_MVdreinulleins() PID_MVeinsnulleins() PID_Pviernulleins()	Berechnung der Stellgrößen (PID-Regler)
TaskSETOUTPUT	4	SETOUTPUT()	Zuweisung von konstanten Ausgangssignalen, Zustandsbeschreibung und Sollwertdefinition des Szenario *Warmhaltung*

Tabelle 6.2.: Umgebungsspezifische Parameter des Regelalgorithmus

Variable	**CoDeSys**	**Jarvis**	**Bemerkung**
FLAG_JARVIS	FALSE	TRUE	Aktivierung/Deaktivierung von entwicklungsumgebungsspezifischem Code
Skalierung_Temp_clamp	10	1	Temperaturen skalieren
OFFSET_Kelvin	0	273,15	Grenzwerte anpassen
Ausgangssignal_min	0	0,004	Min. Ausgangssignal der Steuerung
Ausgangssignal_max	32 767	0,02	Max. Ausgangssignal der Steuerung
Y_MIN aller PID	0	$-0{,}02$	*AnalogDriveController* nur in Jarvis verfügbar
Y_MAX aller PID	32 767	0,02	
KP von pid_MV301	500	$2{,}5 \cdot 10^{-5}$	
TN von pid_MV301	30	$1 \cdot 10^{7}$	
KP von pid_MV101	300	$5 \cdot 10^{-5}$	
TN von pid_MV101	40	$1 \cdot 10^{6}$	
KP von pid_P401	300	$5 \cdot 10^{-5}$	
TN von pid_P401	40	$1 \cdot 10^{6}$	

spezifischen maximalen und minimalen Ausgangssignalen umgesetzt. Eine Anpassung der Regelparameter war aus dem gleichen Grund ebenfalls notwendig. Die PID-Regler der Entwicklungsumgebung *CoDeSys 2.3* generieren ein Ausgangssignal im Regelbereich von 0 bis 32 767, wohingegen die Regler der Simulationsumgebung ein Signal im Regelbereich von 0,004 bis 0,02 erzeugen. Demnach müssen die *KP*- und *TN*-Werte in unterschiedlichen Größenordnungen liegen.

Die Regelparameter der Hardwaretestumgebung wurden so eingestellt, dass die Veränderung der Aktoren bei Änderungen von Sollwerten oder Systemzuständen visuell erkennbar ist. Die Einstellungen der Regelparameter der Simulationsumgebung sind in Kapitel 5 beschrieben. Codezeilen, die nur in einer der beiden Umgebungen aufgerufen werden sollen, können durch *FLAG_JARVIS* und *IF-Abfragen* definiert werden. Dies wird anhand des folgenden Beispiels verdeutlicht.

In der Hardwaretestumgebung wird die Netztemperatur nicht gemessen. Sie entspricht der Netzsolltemperatur, die abhängig von der Außentemperatur berechnet wird (siehe Kapitel 4.3.2). In der Simulationsumgebung wird die Netztemperatur durch die Komponente *generator* eingestellt und vom Temperatursensor *temp_sensor_help* eingelesen. Dies wurde im Anwenderprogramm *PLC_PRG_TWW()* wie in Quellcode 6.1 ersichtlich umgesetzt.

Quellcode 6.1: Quellcode zur Definition der Netztemperatur

```
1  ...
2  IF FLAG_JARVIS = TRUE THEN
3   T_Netz_ist := WORD_TO_REAL(temp_sensor_help_clamp)/Skalierung_Temp_clamp;
4  END_IF;
5  ...
6  T_Netz_soll := 70 + OFFSET_Kelvin;
7  IF FLAG_JARVIS = FALSE THEN
8   T_Netz_soll := T_Netz_start - (T_aussen/2);
9   T_Netz_ist := T_Netz_soll;
10 END_IF;
11 ...
```

Ähnliche Abhängigkeiten sind in der folgenden Aufzählung beschrieben.

Leistungsberechnung In der Hardwaretestumgebung wird die abgenommene Leistung wie in Kapitel 4.3.2 beschrieben berechnet. In der Simulationsumgebung ändert sich die Berechnung des Massenstroms aufgrund des veränderten Regelbereichs.

Berechnung des Massenstroms Die Massenströme für die Funktionsprüfung und die Inbetriebnahme werden wegen der unterschiedlichen Regelbereiche auf zwei verschiedene Arten berechnet.

Regelung von MV302 In der Hardwaretestumgebung ist der Aktor *MV302* als Drei-Wege-Ventil in Form eines Amperemeters umgesetzt. Das anliegende Steuersignal beträgt

entweder 0,004 mA oder 0,02 mA. In der Simulationsumgebung wird das Drei-Wege-Ventil von einer Pumpe simuliert. Diese Pumpe fördert entweder den gleichen Massenstrom wie Pumpe *P401* oder gar keinen.

6.3. PID-Regler

Die PID-Regler werden in *CoDeSyS 2.3* zyklisch von der Task *TaskPID* aufgerufen und sind Bausteine vom Typ *Programm* in der Programmiersprache *FUP*. Die Parameter, die unabhängig von der Entwicklungsumgebung, also für Hardware- und Simulationsumgebung gleichermaßen gelten, sind in Tabelle 6.3 aufgelistet.

6.4. Regelbeschreibung

In der Regelbeschreibung wird zwischen den Szenarien der iWÜST mit TWW und den Szenarien der iWÜST ohne TWW unterschieden. In den folgenden Aufzählungen ist die Szenarionummer (Sz.-Nr.) des jeweiligen Szenarios in Klammern angegeben. Diese Nummer entspricht der Priorität des Szenarios. In den Tabellen 6.4 (Seite 84) und 6.5 (Seite 85) sind die Grenzwerte für die Aktivierung (“aktiv”) und die Deaktivierung (“deaktiv”) sowie die Sollwerte der jeweiligen Szenarien im Bezug auf das für die Validierung verwendete Modell zusammengefasst. Szenarien, die nicht in der Tabelle aufgeführt sind, werden durch ein Signal von der Leitwarte ausgelöst und deaktiviert.

Tabelle 6.3.: Parametrierung der PID-Regler

Parameter	**pid_MV301**	**pid_MV101**	**pid_P401**	**pid_generator**
ACTUAL	MV301_ist	MV101_ist	TIRC401_clamp_	T_Netz_ist
SET_POINT	MV301_soll	MV101_soll	TIRC401_clamp_soll	T_Netz_soll
TV	0	0	0	0
Y_MANUAL	0	0	0	0
Y_OFFSET	0	0	0	0
MANUAL	FALSE	FALSE	FALSE	FALSE
RESET	FALSE	FALSE	FALSE	FALSE
Y	MV301_	MV101_	P401_	generator_
Aktor	MV301	MV101	P401	generator

6.4.1. iWÜST mit TWW

Für ein stabiles und netzdienliches Lademanagement des Warmwasserspeichers wurden Szenarien und zugehörige Regelstrategien beschrieben, die in der folgenden Aufzählung erläutert werden.

Normalbetrieb (10) Der Sollwert des SOC wird von der Leitwarte vorgegeben. Dieser Wert soll vom PID-Regler *pid_MV301* eingestellt werden. Der Regler *pid_P401* soll die Pumpe *P401* regeln, sodass der festgelegte Sollwert von 60 °C am Temperatursensor *TIRC401* erreicht wird.

Rücklauftemperaturbegrenzung (9) Die Rückführung des Wärmeträgermediums mit einer hohen Temperatur in den Rücklauf des Wärmenetzes soll durch dieses Szenario verhindert werden. Durch das Überschreiten einer Grenztemperatur *TIRC302_max* am Temperatursensor *TIRC302* wird das Szenario ausgelöst. Die Rücklauftemperaturbegrenzung wird deaktiviert, sobald die Rücklauftemperatur am Sensor *TIRC302* unter den Grenzwert *TIRC302_zulaessig* gesunken ist. Ist das Szenario aktiviert, so wird der Sollwert des SOC auf 10 % (*SOC_RL_Begrenzung*) gesetzt. Dadurch wird der Speicher entladen und das Wärmeträgermedium aus dem primärseitigen Rücklauf nicht in das Wärmenetz, sondern in den Speicher geleitet.

Leistungsbegrenzung (8) Der maximal zugelassene Leistungsbezug des Kundens ist vertraglich festgelegt. Bezieht der Kunde über einen Zeitraum von 15 Minuten im Mittel eine Leistung, die die vereinbarte Leistung unter Berücksichtigung einer Kulanz übersteigt, soll die Leistungsbegrenzung aktiviert und die Leistungsabnahme durch die Reduzierung des Speicherfüllstands verringert werden (*SOC_soll* = *SOC_min* = 33 %). Durch das Verringern des SOC wird der Speicher geleert und kein Massenstrom vom Aktor *MV301* in das Wärmenetz geleitet. Das Szenario wird dektiviert, sobald die mittlere Leistungsabnahme unter der vereinbarten maximalen Leistung (ohne Kulanz) liegt.

Füllstandsbegrenzung (7) Das Szenario *Füllstandsbegrenzung* wird ausgelöst, sobald die Temperaturdifferenz zwischen Netztemperatur und Rücklauftemperatur des Temperatursensor *TIRC305* kleiner als *T_delta_Fuellstandsbegrenzung_in* ist und verhindert ein Überladen des Speichers. Das Szenario wird deaktiviert, sobald die Temperaturdifferenz größer als *T_delta_Fuellstandsbegrenzung_out* ist und die durch den Timer *Timer_Fuellstandsbegrenzung* vorgegebene Dauer von 8 Minuten vergangen ist.

Speicher Überladen (6) Das Szenario *Speicher Überladen* wird in der Realität extern von der Leitwarte durch Setzen der Variable *EXTERN_Fuellstand_max_aktiv* auf *TRUE* aktiviert. Der Sollspeicherfüllstand beträgt dann 100 % (*SOC_max*). Der Wärmespeicher wird maximal gefüllt und der Netzrücklauf mögichst aufgeheizt. Dieses Szenario kann bspw. bei einer Überproduktion von regenerativer Wärme aktiviert werden.

Funktionsprüfung (5) Die Funktionsprüfung wird von der Leitwarte durch das Setzen der Variable *EXTERN_DO_funktionspruefung_TWW* auf *TRUE* gestartet, sofern der SOC des Speichers über dem festgelegten Minimalwert (*SOC_freigabe_funktionspruefung*) liegt. Anschließend wird die Funktionalität der Durchgangsventile des gesamten Systems geprüft.

Netzkollaps (4) Das Szenario *Netzkollaps* wird durch Setzen der Variable *EXTERN_Netzkollaps* auf *TRUE* ausgelöst und durch Zurücksetzen auf *FALSE* deaktiviert. Ist das Szenario aktiv, so wird dem Sollwert des SOC (*SOC_soll*) der Wert *SOC_min* zugewiesen.

Kommunikationsausfall (3) Die Leitwarte sendet in einem definierten Abstand abwechselnd ein sogenanntes *Toggle*-Signal an die Steuerung der iWÜST. Bei dem Signal handelt es sich um eine Variable, deren Wert entweder *TRUE* oder *FALSE* ist. Anhand diesem Signal wird die Verbindung der iWÜST zur Leitwarte vom Regelalgorithmus festgestellt. Bleibt das Signal für einen Zeitraum von 15 Minuten unverändert, so interpretiert die Steuerung dies als Kommunikationsausfall (*SOC_soll = SOC_min*).

Minimale Temperatur (2) Hinsichtlich der Versorgung der Brauchwasseranschlüsse soll gewährleistet sein, dass die Solltemperatur am Sensor *TIRC401* gehalten wird. Dies wird sichergestellt, indem der Aktor *MV301* nicht wie sonst auf einen Sollwert des SOC geregelt wird, sondern auf den Sollwert *TIRC324_soll* der Temperatur am Sensor *TIRC324*. Wenn die Temperaturen *TIRC324* und *TIRC323* unter die ihnen zugewiesenen Grenzwerte fallen wird das Szenario ausgelöst. Sobald die Temperaturen des obersten Speichersensors *TIRC324* und des mittleren Speichersensors *TIRC322* einen Grenzwert überschritten haben wird das Szenario deaktiviert.

Inbetriebnahme (1) Dieses Szenario wird zur Inbetriebnahme der Anlage nach Installation, Wartung oder Reperatur durchgeführt. Dabei werden verschiedene Hard- und Softwaretests ausgelöst, welche die vollständige Funktionalität der Anlage verifizieren. Während der Inbetriebnahme wird die Nummer des aktuellen Tests (*Test-Nr.*) der Variable *Inbetriebnahme_Szenario_TWW* zugewiesen, welche von der Leitwarte ausgelesen werden kann.

6.4.2. iWÜST ohne TWW

Für eine hohe Versorgungssicherheit und eine netzdienlichen Fahrweise wurden die folgenden Regelstrategien für verschiedene Systemzustände der iWÜST ohne TWW definiert.

Tabelle 6.4.: Parameter der Regelung der iWÜST mit TWW (hellgrau: Netztemperatur < 75 °C, dunkelgrau: Netztemperatur > 75 °C)

Sz.-Nr.	Spezifikation	Name	Wert
10	SOC_soll	SOC_Leitwarte	80 %
	Soll-Temp.	TIRC401_soll	60 °C
9	aktiv	TIR302_max	55 °C
	deaktiv	TIR302_zulaessig	50 °C
	SOC_soll	SOC_RL_Begrenzung	10 %
8	aktiv	P_WMZ001_max	115,5 kW
	deaktiv	P_WMZ001_zulaessig	105 kW
	Kulanzfaktor	f_ueberschreitung_zulaessig	1,1
	SOC_soll	SOC_min	33 %
7	aktiv	T_delta_Fuellstandsbegrenzung_in	10 °C
	deaktiv	T_delta_Fuellstandsbegrenzung_out	15 °C
	SOC_soll	SOC_Fuellstandsbegrenzung	50 %
	min. Öffnung MV301	MV301_offen_min	5 %
5	Freigabe	SOC_freigabe_funktionspruefung_in	20 %
	max. Massenstrom	Massenstrom_max	2,4 kg/s
	min. Massenstrom	Massenstrom_min	0 kg/s
4	SOC_soll	SOC_Stoerfall-inCodeändern	10 %
3	SOC_soll	SOC_Stoerfall-inCodeändern	10 %
2	aktiv TIRC324	TIRC324_min	65,15 °C 70,15 °C
	aktiv TIRC323	TIRC32X_grenz	TIRC401_soll + 3 °C
	deaktiv TIRC324	TIRC324_ladung_ausstieg	68,15 °C 74,15 °C
	deaktiv TIRC322	TIRC32X_grenz	TIRC401_soll + 3 °C
	TIRC324_soll	TIRC324_soll_kopfraum	70,15 °C 76,15 °C

Tabelle 6.5.: Parameter der Regelung der iWÜST ohne TWW

Sz.-Nr.	Spezifikation	Name	Wert
10	Soll-Temp.	TIRC101_soll	70 °C
9	aktiv	TIR103_max	55 °C
	deaktiv	TIR103_zulaessig	40 °C
	deaktiv	dT_RLbegrenzung_grenz	20 °C
8	aktiv	P_WMZ001_max	115,5 kW
	deaktiv	P_WMZ001_zulaessig	105 kW
	Kulanzfaktor	f_ueberschreitung_zulaessig	1,1
2	aktiv	MV101_offen_min	3 %
	deaktiv	MV101_offen_zulaessig	5 %

Normalbetrieb (10) Der Normalbetrieb ist das am niedrigsten priorisierte Szenario und dementsprechend automatisch aktiviert, wenn keines der anderen Szenarien ausgelöst wurde. Der PID-Regler *pid_MV101* regelt den Aktor *MV101* so, dass sich am Temperatursensor *TC101* möglichst der Temperatursollwert *TC101_soll* einstellt.

Rücklaufbegrenzung(9) Bei geringer Leistungsabnahme im Wasserkreislauf *Heizkreis* kann das Wärmeträgermedium im primären Rücklauf des Wärmeübertragers eine hohe Temperatur erreichen. Die Rücklaufbegrenzung wird ausgelöst, sobald diese Temperatur, welche am Sensor *TIRC103* gemessen wird, höher als der Grenzwert *TIRC103_max* ist. Wenn dieser Fall eintritt, soll der Aktor *MV101* so geregelt werden, dass die Temperatur am Sensor *TIRC103* verringert wird.

Leistungsbegrenzung (8) Die Leistungsbegrenzung wird ausgelöst, sobald die Leistungsabnahme der letzten 20 Minuten im Mittel die vertraglich vereinbarte Leistung mit Kulanz überschreitet. Das Szenario soll deaktiviert werden, wenn die Leistungsabnahme der letzten 20 Minuten im Mittel unter der vertragliche vereinbarten Grenze liegt. Sobald das Szenario ausgelöst wurde, wird der Sollwert *TC101_soll* im Abstand von 30 Sekunden um 1 °C verringert, bis das Austrittskriterium erfüllt ist. Ein Timer sorgt dafür, dass die Leistungsbegrenzung bei einer konstanten Leistungsabnahme im Bereich des Grenzwerts nicht in kurzen Abständen aktiviert und deaktiviert wird.

Rücklauftemperaturregelung (6) Es soll eine definierte Rücklauftemperatur eingestellt werden, sobald die Aktivierung der Rücklauftemperaturregelung durch die Leitwarte erfolgt ist (*EXTERN_RLTempRegelung = TRUE*).

Funktionsprüfung (5) Die Prüfung der Funktionalität des Ventils *MV101* wird von der Leitwarte durch das Setzen der Variable *EXTERN_DO_funktionspruefung_WUE* auf *TRUE* aktiviert.

Netzkollaps (4) Der Netzkollaps wird durch das Setzen der Variable *EXTERN_Netzkollaps* von der Leitwarte auf *TRUE* ausgelöst und durch Setzen auf *FALSE* deaktiviert. Der Sollwert *TC101_soll* ist nach der Aktivierung des Szenarios nicht mehr außentemperaturabhängig. Er wird entweder von der Leitwarte vorgegeben (*TC101_Netzkollaps_soll*) oder abhängig von der Netztemperatur und einer vorgegebenen Variable für die Grädigkeit (*T_WUT_graedigkeit*) berechnet.

Kommunikationsausfall (3) Der Kommunikationsausfall wird vom Regelalgorithmus der iWÜST mit TWW detektiert. Der Temperatursollwert *TC101_soll* wird wie beim Netzkollaps festgelegt.

Warmhaltung (2) Die Regelung soll ein Auskühlen der primärseitigen Leitungsabschnitte verhindern, wenn keine oder eine sehr geringe Last abgenommen wird. Eine minimale Ventilöffnung bedingt einen stetigen Volumenstrom in den Leitungen. Das Szenario wird ausgelöst, sobald das Stellsignal des Reglers *pid_MV101* an die Pumpe *MV101* einem Öffnungswinkel entspricht, der kleiner als *MV101_offen_min* ist. Das Stellsignal *MV101_* an die Pumpe *MV101* wird bei aktiviertem Szenario so verändert, dass dieses einem Öffnungwinkel von *MV101_offen_zulaessig* gleichkommt. Das Szenario wird deaktiviert, sobald das Stellsignal *MV101_* des Reglers *pid_MV101* größer als der entsprechendem Öffnungswinkel *MV101_offen_zulaessig* ist.

Inbetriebnahme (1) Äquivalent zur *Inbetriebnahme* der iWÜST mit TWW.

Überdruck- und Übertemperaturbegrenzer (0) Die Überprüfung des Überdruckbegrenzers *TZ* und des Übertemperaturbegrenzers *TS* erfolgt in jedem Zeitschritt. Sobald einer der Sensoren den Wert *TRUE* zurückgibt, wird das Szenario aktiviert und allen Aktoren das minimale Stellsignal gesendet, sodass kein Durchfluss mehr vorhanden ist. Dadurch sollen sicherheitskritische Systemzustände verhindert werden.

6.5. Realisierung der Regelung

Die Regelung wurde in den drei Anwenderprogrammen *PLC_PRG_TWW()*, *PLC_PRG_WUEST()* und *SETOUTPUT()* als Code in der Programmiersprache *ST* umgesetzt. Bei der Programmierung wurde darauf geachtet, dass eine möglichst hohe Diversität bzgl. der verwendeten Funktionen im Code vorhanden ist. Jede verwendete Funktion wird durch den Parser in die Programmiersprache *Python* übersetzt, wodurch die Funktionalität des Parsers hinsichtlich dieser Funktionen validiert wird.
Die Anwenderprogramme *PLC_PRG_TWW()* und *PLC_PRG_WUEST()* sind bzgl. des strukturellen Aufbaus ähnlich und können für einen ersten Überblick gemeinsam beschrieben werden. In Abbildung 6.2 auf Seite 88 ist der Programmablaufplan für beide Anwenderprogramme vereinfacht dargestellt.

Zunächst werden die Temperaturen der Temperatursensoren eingelesen und skaliert. Anschließend werden die festgelegten Variablen für die Grenzwerte aus den Tabellen 6.4 und 6.5 bei dem ersten Programmaufruf der Anwenderprogramme skaliert (siehe Kapitel 6.2). Danach wird überprüft, ob beim aktuellen Systemzustand ein Szenario aktiviert ist, indem die systeminternen Temperaturen und die externen Signale von der Leitwarte mit den Vergleichswerten (Tabelle 6.4 und 6.5) abgeglichen werden. Die Szenarien sind bezüglich ihrer Priorität aufsteigend nummeriert. Werden mehrere Szenarien gleichzeitig ausgelöst wird nur das höher priorisierte berücksichtigt. Dies ist durch die Reihenfolge der einzelnen Abfragen zu den Szenarien im Code der Anwenderprogramme umgesetzt. Gegebenenfalls werden daraufhin Sollwerte den jeweiligen PID-Regler zugewiesen. Abschließend werden spezifische Abfragen und Zuweisungen der einzelnen Anwenderprogramme durchgeführt.

Die Regelbeschreibung für die iWÜST mit TWW wurde im Anwenderprogramm *PLC_PRG_TWW()* umgesetzt. Im Anwenderprogramm *PLC_PRG_WUEST()* ist der Regelalgorithmus der iWÜST ohne TWW (außer *Warmhaltung*) enthalten. Die *Warmhaltung* wurde im Anwenderprogramm *SETOUTPUT()* implementiert, damit diese nach der Berechnung des Ausgangssignals *MV101_* vom Regler *pid_MV101* aufgerufen wird.

Die programmiertechnische Umsetzung der Regelbeschreibung wird im Folgenden exemplarisch anhand der *Rücklauftemperaturbegrenzung* der iWÜST mit TWW erläutert. In der Export-Datei im Anhang A.4 kann die programmiertechnische Realisierung der anderen Szenarien eingesehen werden.

Der Quellcode 6.2 wird alle 200 ms aufgerufen. Wenn die Temperatur am Sensor *TIRC302* größer als *TIRC302_max* (55 °C, siehe Tabelle 6.4) ist oder das Flag *FLAG_9_TWW* den Wert *TRUE* enthält wird dem SOC-Sollwert *SOC_soll* der Wert *SOC_RL_Begrenzung* (10 %) zugewiesen. Der Wert der Variable *Szenario_TWW* zeigt das aktuell aktivierte Szenario an und entspricht 9, wenn das Szenario aktiviert ist. Sobald die Temperatur am Sensor *TIRC302* unter den Wert *TIRC302_zulaessig* (40 °C) sinkt wird das Flag *FLAG_9_TWW* auf *FALSE* gesetzt. Die *Rücklauftemperaturregelung* ist ab diesem Zeitpunkt nicht mehr aktiviert. Um zu vermeiden, dass das Szenario mit einer hohen Frequenz aktiviert und deaktiviert wird, unterscheiden sich die Grenzwerte um 5 °C (Hysterese).

Quellcode 6.2: Quellcode der Rücklauftemperaturbegrenzung

```
1 IF TIRC302 > TIRC302_max OR FLAG_9_TWW = TRUE THEN
2 Szenario_TWW := 9;
3 SOC_soll := SOC_RL_Begrenzung;
4 FLAG_9_TWW := TRUE;
5 END_IF;
6 IF TIRC302 < TIRC302_zulaessig THEN
7 FLAG_9_TWW := FALSE;
8 END_IF;
```

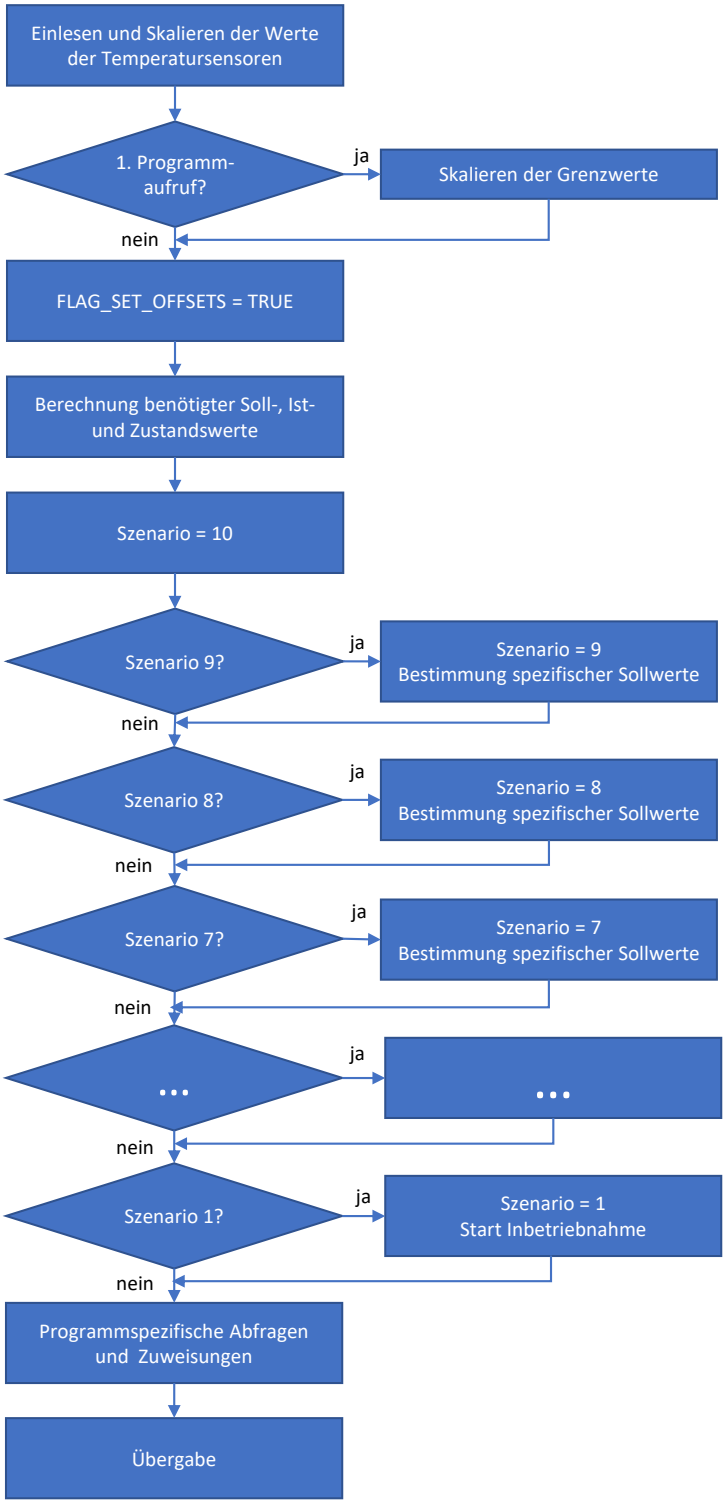


Abbildung 6.2.: Vereinfachter Programmablaufplan der Anwenderprogramme *PLC_PRG_TWW()* und *PLC_PRG_WUEST()*

6.6. Validierung

Der Regelalgorithmus wurde zunächst anhand der Hardwaretestumgebung aus Kapitel 4 getestet. Mit Hilfe der Hardwaretestumgebung und der Entwicklungsumgebung *CoDeSys 2.3* konnten die unterschiedlichen Szenarien ausgelöst und überprüft werden. Anschließend wurde das Steuerungsprogramm mit Hilfe des Parsers aus Kapitel 3 in die Simulationsumgebung *Jarvis* geladen und im Simulationsmodell aus Kapitel 5 getestet und anhand der Simulationsergebnisse validiert. Durch dieses Vorgehen wurden die Hardwaretestumgebung, der CODESYSImporter, der Parser, das Simulationsmodell und der Regelalgorithmus validiert.

6.6.1. Validierung des Regelalgorithmus in der Hardwaretestumgebung

Zur Validierung des Regelalgorithmus in der Hardwaretestumgebung wurde dieser auf die SPS geladen und gestartet. Szenarien, deren Aktivierung temperaturabhängig ist, konnten dann durch die Veränderung des Widerstands der jeweiligen Potentiometer ausgelöst werden. Die Aktivierung und Deaktivierung von leitwartengesteuerten Szenarien konnte ebenfalls simuliert werden, da Variablen in der Entwicklungsumgebung online verändert werden können. Durch die Anzeigen der Amperemeter sowie der Leuchtkraft der LED konnte überprüft werden, ob die geforderte Reaktion der Aktoren von der Steuerung ausgelöst wurden. Dieses Vorgehen wird durch eine Bilderstrecke in Abbildung A.19 im Anhang für die *Rücklauftemperaturbegrenzung* der iWÜST mit TWW verdeutlicht.

Die Validierung offenbarte Schwachstellen in der Umsetzung der Regelbeschreibung, welche durch unterschiedliche Anpassungen des Programmcodes gelöst werden konnten. So hat sich bspw. bei der Validierung der *Leistungsbegrenzung* gezeigt, dass bei einem geringfügigen, konstanten Überschreiten des oberen Leistungsgrenzwerts das Ventil (dargestellt durch das Amperemeter) *MV301* mit einer hohen Frequenz öffnet und schließt, wodurch im realen System Schwingungen entstehen könnten. Aufgrund dessen wurde ein Timer vom Typ *TOF* integriert, der dafür sorgt, dass der Sollwert des SOC nach der Erfüllung des Austrittskriteriums noch für die folgenden zwei Minuten $33\,\%$ beträgt.

Grundsätzlich konnten alle Szenarien anhand der Hardwaretestumgebung validiert werden. Die Reaktion der Aktoren auf die Veränderungen der Widerstandswerte und der Variablen entspricht dem, was in der Regelbeschreibung vorgesehen ist. Es sollte dabei jedoch berücksichtigt werden, dass dem Modell jegliche Trägheitskomponente der realen Anlage fehlt. Dadurch besteht die Möglichkeit, dass Prozesse aufgrund des Modells bzgl. des zeitlichen Verhaltens nicht richtig nachgebildet werden. Eine Aussage bzgl. der Güte des

Regelalgorithmus kann somit in fast allen Fällen nur unter Zuhilfenahme der Simulationsumgebung getroffen werden. Die Hardwaretestumgebung eignete sich lediglich zur Validierung des Szenario *Inbetriebnahme* aufgrund der Fülle an Hardwaretests besser als die Simulationsumgebung.

6.6.2. Validierung des Regelalgorithmus in der Simulationsumgebung

Das iWÜST-Modell aus Kapitel 5 wurde für die Validierung des Regelalgorithmus in der Simulationsumgebung verwendet. Die Parameter der Komponenten entsprechen grundsätzlich jenen aus der *Testsimulation* (siehe Kapitel 5.3). Teilweise wurden jedoch Systemzustände (bspw. die Lastabnahme) so angepasst, dass sichergestellt ist, dass das behandelte Szenario ausgelöst wird bzw. manche Szenarien nicht ausgelöst werden. Die Anpassungen für die jeweiligen Szenarien werden in den Tabellen 6.6 und 6.8 (Seite 101) beschrieben. Die in der Tabelle beschriebenen Zeitpunkte und Zeiträume entsprechen den Markierungen in den Grafiken zur Validierung der Szenarien in den jeweiligen Abschnitten. In einigen Fällen werden neben dem behandelten Szenario weitere Szenarien aufgrund der Systemzustände ausgelöst. Darauf wird jedoch bei der Validierung nicht weiter eingegangen.
Die *Testsimulation*, mit der die Validierung der Szenarien durchgeführt wurde, startet zu dem willkürlich gewählten Zeitpunkt 14:02 Uhr und endet nach 85 Minuten. In den folgenden Abbildungen 6.3 bis 6.23 werden die Ergebnisse erst ab dem Zeitpunkt 14:07 Uhr dargestellt, da sich das System in den ersten Minuten nach Simulationsstart einschwingt. Dabei werden durch Initaltemperaturen in den Komponenten und Rohrleitungen irrtümlich Szenarien ausgelöst, welche jedoch keinen Wert für die Validierung haben.
Im Folgenden wird zunächst der Regelalgorithmus für die einzelnen Szenarien der iWÜST mit TWW validiert. Anschließend folgt die Validierung des Regelalgorithmus der iWÜST ohne

Tabelle 6.6.: Veränderungen in der Parametrierung für die Validierung der jeweiligen Szenarien der iWÜST mit TWW

Sz.-Nr.	Parameter	Zeitraum	Wert
8	*load1*	t_1 - Simulationsende	70 kW
7	min. Lastabnahme *load1*	gesamter Simulationszeitraum	8 kW
	SOC_soll	gesamter Simulationszeitraum	90 %
6	SOC_soll	gesamter Simulationszeitraum	80 %
3	EXTERN_Fuellstand_max_aktiv	t_1 - t_2	*TRUE*
2	T_Netz_soll	gesamter Simulationszeitraum	80 °C
	SOC_soll	t_1 - Simulationsende	40 %

TWW. Die Netztemperatur beträgt bei den Simulationen der iWÜST mit TWW 70 °C, bei der iWÜST ohne TWW 90 °C.
Die allgemeine Vorgehensweise wird anhand der Validierung der *Rücklauftemperaturbegrenzung* deutlich, welche im Folgenden ausführlich erläutert wird. In den darauf folgenden Abschnitten wird auf eine ausführliche Analyse verzichtet. Diese kann im Anhang A für die einzelnen Szenarien nachvollzogen werden. Das Szenario *Normalbetrieb* wird in diesem Kapitel nicht behandelt, da die Regelungsparametrierung aus Kapitel 5.3 unter den Voraussetzung des *Normalbetriebs* durchgeführt wurden. Die Simulationsergebnisse zeigen, dass die Regelbeschreibung bzgl. der Temperatur- und SOC-Regelung erfüllt wird, siehe z.B. Abbildung 5.13.

Rücklauftemperaturbegrenzung (iWÜST mit TWW)

Die *Rücklauftemperaturbegrenzung* wird zum Zeitpunkt t_1 in Abbildung 6.3 ausgelöst, da die Temperatur *TIRC302* den Grenzwert *TIRC302_max* überschreitet. Der Temperaturverlauf am Sensor *TIRC302* wird durch die Lastabnahme der Last *load2* bestimmt. Das Szenario ist so lange aktiviert, bis die Rücklauftemperatur unter den Wert *TIRC302_zulaessig* fällt (t_2). Ab der Aktivierung des Szenarios wird von der Pumpe *MV301* kein Wärmeträgermedium in das Wärmenetz gefördert (siehe Abbildung 6.4). Dadurch steigt die Temperatur am Sensor *TIRC302* nicht weiter an, wodurch eine höhere Rücklauftemperatur verhindert werden kann. Das Verhalten entspricht somit dem gewünschten Verhalten aus der Regelbeschreibung.

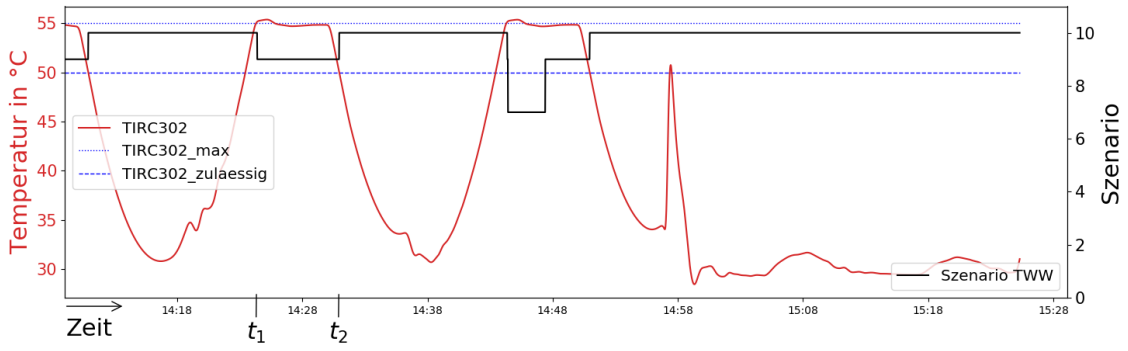


Abbildung 6.3.: Rücklauftemperaturbegrenzung (iWÜST mit TWW, TIRC302)

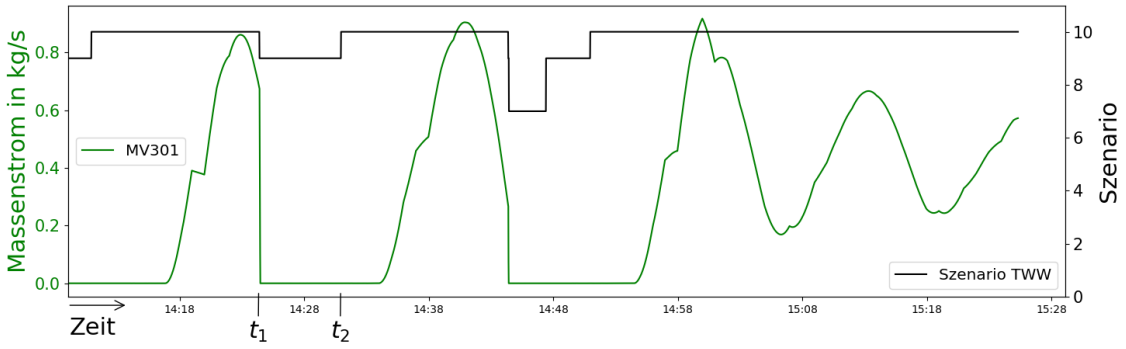


Abbildung 6.4.: Rücklauftemperaturbegrenzung (iWÜST mit TWW, MV301)

Leistungsbegrenzung (iWÜST mit TWW)

In Abbildung 6.5 ist der zeitliche Verlauf der 15-Minuten-Mittelwerte der Leistungsabnahme mit dem aktivierten Szenario gezeigt. Bei Überschreitung des Grenzwerts zum Zeitpunkt t_2 wird das Szenario ausgelöst. Die untere Grenze der *Leistungsbegrenzung* wird zum Zeitpunkt t_3 unterschritten und das Szenario ist ab diesem Zeitpunkt nicht mehr aktiviert. Die grafisch dargestellten Simulationsergebnisse machen deutlich, dass die Anforderungen aus der Regelbeschreibung durch den Regelalgorithmus erfüllt werden. Durch das Auslösen des Szenarios wird die Leistungsabnahme verringert, bis der Mittelwert unter dem vertraglich vereinbarten Grenzwert liegt.

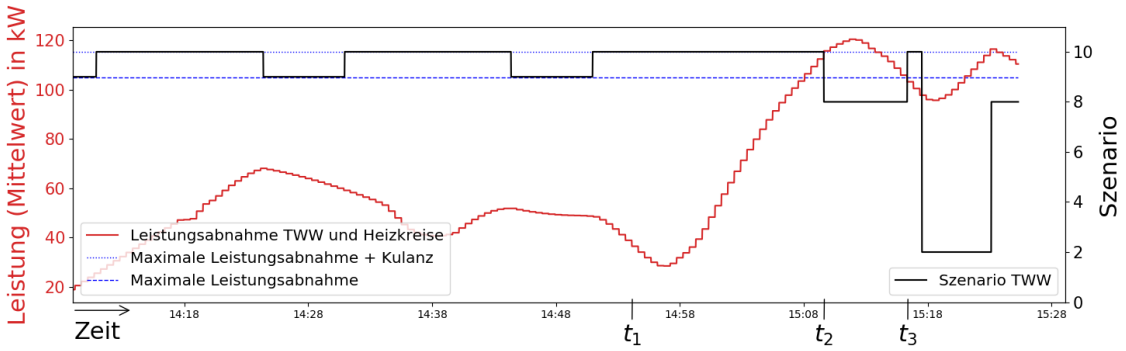


Abbildung 6.5.: Leistungsbegrenzung (iWÜST mit TWW)[1]

[1] t_1 - Simulationsende: Leistungsabnahme *load1* = 70 kW

Füllstandsbegrenzung (iWÜST mit TWW)

Die erste Analyse der Simulationsergebnisse zur Validierung der *Füllstandsbegrenzung* zeigte, dass bei der Berechnung der Temperaturen in der Komponente *pipe8* ein Fehler im Modellcode der Simulationsumgebung vorliegt. Daraufhin wurde die Komponente *pipe8* aus dem Modell entfernt. Des Weiteren musste der Regelalgorithmus geringfügig aufgrund der regelungstechnischen Modelleigenschaften (Überschwingen der Netztemperatur bei schnellem Schließen des Ventils *MV301*) angepasst werden. Eine detaillierte Beschreibung ist dazu im Anhang (Seite LXXVIII) zu finden. Diese Änderungen haben keinen Einfluss auf die Ergebnisse der Validierung.
Die Simulationsergebnisse unter Verwendung des angepassten Modells sind in Abbildung 6.6 dargestellt. Nach Unterschreiten des Grenzwerts der Temperaturdifferenz (Eintrittskriterium)

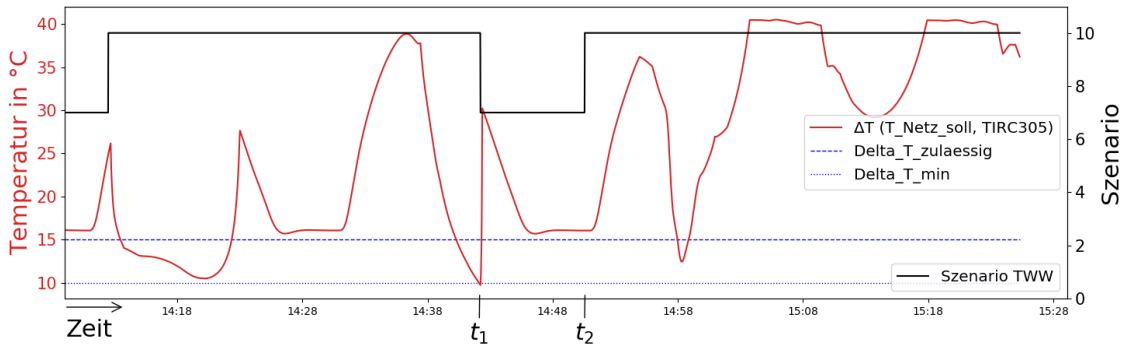


Abbildung 6.6.: Füllstandsbegrenzung (iWÜST mit TWW)

wird die *Füllstandsbegrenzung* zum Zeitpunkt t_1 aktiviert. Die Temperaturdifferenz steigt nach Aktivierung des Szenarios auf 30 °C an und liegt damit über dem Grenzwert zur Deaktivierung des Szenarios. Das liegt daran, dass ab dem Zeitpunkt des Szenarioeintritts nicht mehr das Wärmeträgermedium aus dem Speicher, sondern das kältere Medium aus dem Rücklauf des Wärmeübertragers in den Rücklauf des Wärmenetzes fließt. Im weiteren Verlauf verändert sich die Temperatur am Sensor *TIRC305* durch die Lastabnahme von *load2*. Zum Zeitpunkt t_2 wird das Szenario nach Ablauf der Wartezeit durch den Timer (8 Minuten) deaktiviert.
Das Austrittskriterium dieses Szenarios hat nicht den gewünschten Effekt bzgl. der Füllstandsbegrenzung. Zunächst wird nun der Fall beschrieben, wenn eine Rohrleitung zwischen dem T-Stück *t_piece6* und dem Sensor *TIR305* vorhanden wäre. Durch die Rohrleitung vor dem Sensor würde die Temperatur, die zum Auslösen des Szenarios geführt hat, über einen längeren Zeitraum im Rohr vorhanden sein, da durch den geringen Massenstrom viel Volumen vom Wärmeträgermedium dieser Temperatur im Rohr vorhanden ist. Nach einer bestimmten, von der Länge des Rohrs abhängigen Zeit würde der Sensor eine sprunghafte Temperaturänderung erfahren. Dabei würde nun die Rücklauftemperatur aus dem Wärme-

übertrager gemessen werden, wodurch das Szenario deaktiviert wird. Bei sehr kurzer (im Modell: keiner) Rohrleitung wird am Sensor *TIRC305* sehr schnell die Rücklauftemperatur des Wärmeübertragers gemessen. Durch den Timer *Timer_Fuellstandsbegrenzung* wird der Speicher zwar für die folgenden 8 Minuten entleert, jedoch steht die Temperatur *TIRC305* nach Eintreten des Szenarios, unabhängig von der Existenz einer Rohrleitung (*pipe8*), in keinem Zusammenhang mit dem Füllstand des Speichers. Für die Begrenzung des Füllstands wird demnach eine andere Strategie benötigt. Denkbar wäre hier die Berücksichtigung der untersten Speichertemperatur *TIRC320*. Des Weiteren könnte bei einem häufigeren Auslösen des Szenarios in einem bestimmten Zeitraum der Sollwert des SOC (*SOC_Leitwarte*) verringert werden.

Speicher Überladen (iWÜST mit TWW)

Anhand Abbildung 6.7 kann gezeigt werden, dass die Anforderungen der Regelbeschreibung an das Szenario *Speicher Überladen* vom Regelalgorithmus wie gefordert erfüllt werden. Der Speicher wird durch das Auslösen des Szenarios (t_1) komplett gefüllt. Das Aufheizen des Netzrücklaufs ist anhand des Temperaturverlaufs des Sensors *TIRC305* in Abbildung 6.8 ersichtlich.

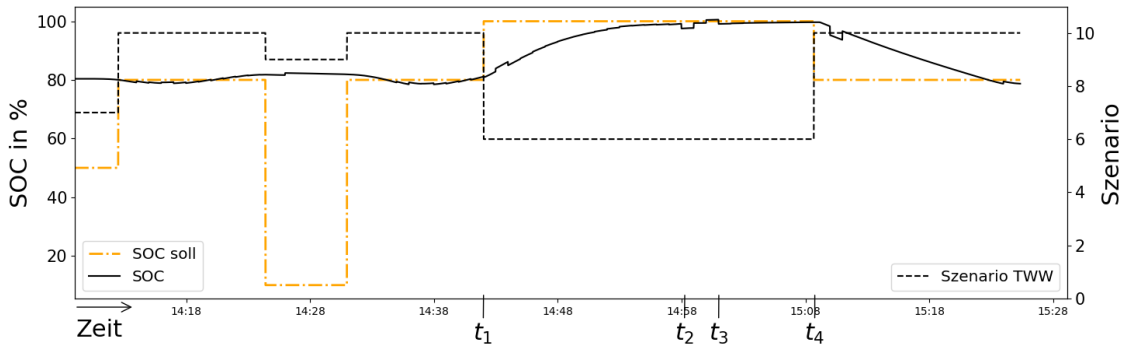


Abbildung 6.7.: Speicher Überladen (iWÜST mit TWW, SOC)[2]

Bei Austritt aus dem Szenario (t_4) würde sofort Szenario 7 aktiviert werden, da der Netzrücklauf aufgeheizt wurde. Infolgedessen würde der Sollwert des SOC 50 % betragen. In Situationen, in denen das Netz aufgrund von Überschüssen aufgeheizt werden soll, wäre ein anschließendes Entladen des Speichers nicht netzdienlich, da grundsätzlich davon ausgegangen werden kann, dass in diesen Situationen selbst nach Deaktivierung des Szenarios ausreichend Leistung für die Wärmeerzeugung vorhanden ist. Aus diesem Grund wurde ein

[2] t_2: Netz-Temp. sinkt kurzzeitig auf 69,2 °C, t_3: kurzzeitig SOC > 100% wegen geringer Netztemperatur

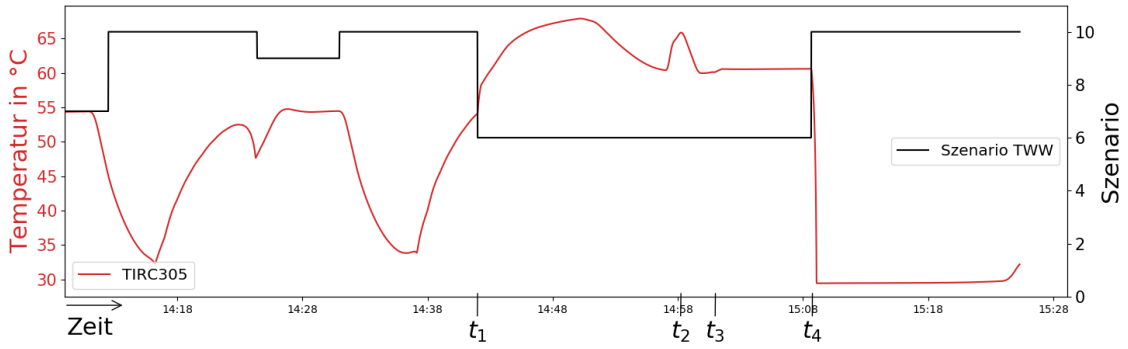


Abbildung 6.8.: Speicher Überladen (iWÜST mit TWW, TIRC305)[2]

Timer in das Programm integriert, der die Aktivierung des Szenario 7 für 8 Minuten nach der Deaktivierung des Szeanrio 6 blockiert. So ist sichergestellt, dass das Szenario 7 auch nicht reaktiviert wird, falls es vor Aktivierung von Szenario 6 in Betrieb war und aufgrund des Timers *Timer_Fuellstandsbegrenzung* noch nicht deaktiviert wurde.

Funktionsprüfung (iWÜST mit TWW und iWÜST ohne TWW)

In diesem Abschnitt wird die Validierung der *Funktionsprüfung* für beide iWÜST durchgeführt. Die Ventile der beiden iWÜST können nicht einzeln geprüft werden, da der Volumenstrom vom Wärmemengenzähler nur für die gesamte Anlage gemessen wird. Sobald das Szenario der iWÜST mit TWW aktiviert ist, wird die *Funktionsprüfung* der iWÜST ohne TWW ebenfalls gestartet und gleichzeitig durchgeführt. Das maximale und minimale Stellsignal wird in der Tasks *SETOUTPUT()* nach dem Code des Szenario *Warmhaltung* zugewiesen, sodass kein minimaler Öffnungswinkel vorhanden ist und das Ausgangssignal der PID-Regler überschrieben wird. Die Ergebniscodes der Funktionsprüfung sind in Tabelle 6.7 aufgeführt.

Da in den verwendeten Modellen der iWÜST keine Möglichkeit besteht, einen Volumenstrom zu erfassen, wurde für die Validierung des Szenarios der Volumenstrom mit Hilfe der Steuersignale, die an die Aktoren geschickt werden, simuliert (siehe Formel 4.2). Die Steuersignale werden von dem Regelalgorithmus vorgegeben, weshalb diese innerhalb eines Zeitschritts an-

Tabelle 6.7.: Parameter und Ergebniscodes der Funktionsprüfung

Ergebniscode	Bedeutung
200	Funktionsprüfung erfolgreich
4001	Prüfung min.Massenstrom fehlgeschlagen
4002	Prüfung max. Massenstrom fehlgeschlagen

liegen und die simulierte Prüfung somit abgeschlossen ist. In der Realtät würde das Verfahren der Ventile auf die maximale und minimale Stellung einige Sekunden in Anspruch nehmen. Weiterhin gibt es in der Simulation und der Hardwaretestumgebung im hydraulischen Sinn keine defekten Bauteile. Daher wurde ein Timer (30 s) implementiert, sodass die einzelnen Schritte der Funktionsprüfung in einer grafischen Auswertung sichtbar gemacht werden können. Es kann dadurch zumindest die korrekte Abfolge der vorgesehenen Schritte sowie die richtige Interpretation der Werte geprüft werden.
In Abbildung 6.9 sind die Massenströme der Pumpen *MV101* und *MV301*, der gesamte Massenstrom der Anlage sowie das Szenario im zeitlichen Verlauf dargestellt. Zum Zeitpunkt der Aktivierung des Szenarios (t_1) fördern die Pumpen 30 s lang keinen Massenstrom. Anschließend wird für 30 s der maximale Massenstrom gefördert, wodurch ein Gesamtmassenstrom von 2,4 kg/s vom simulierten Wärmemengenzähler gemessen wird. Ist die Funktionsprüfung nach einer Minute abgeschlossen, wird sie wieder von neuem gestartet, bis sie von der Leitwarte beendet wird (t_2). Der Ergebniscode der Funktionsprüfung wurde mit *200* korrekt ausgegeben. Somit ist die Funktionalität des Szenarios im Bereich des Möglichen mit den vorhandenen Modellen validiert.

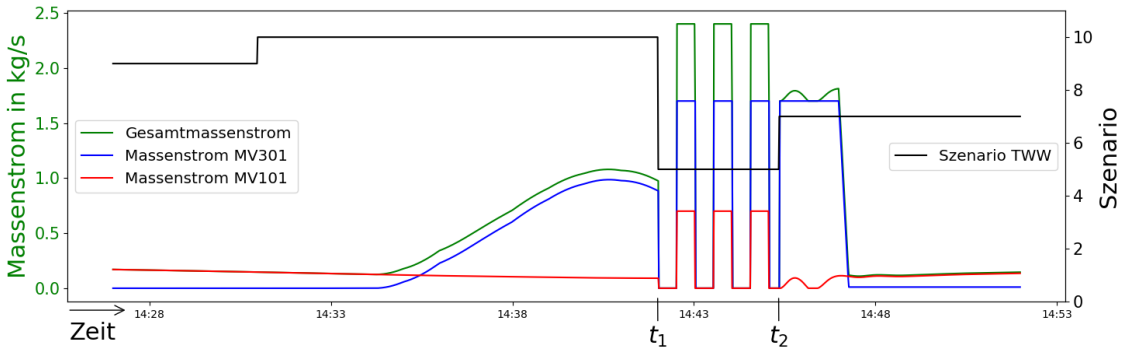


Abbildung 6.9.: Funktionsprüfung (iWÜST mit TWW)

Netzkollaps (iWÜST mit TWW)

Anhand der Abbildungen 6.10 und 6.11 wird die Validierung des Szenario *Netzkollaps* durchgeführt. Der SOC sinkt ab dem Eintritt in das Szenario (Abbildung 6.10, Zeitpunkt t_1) so lange (*SOC_soll* = 10 %), bis die Temperaturen im oberen Bereich des Speichers zu gering werden und das Szenario *Minimale Temperatur* aktiviert wird (t_2). Beim Szenario *Minimale Temperatur* ist die Stellgröße des Reglers *pid_MV301* die oberste Speichertemperatur *TIRC324*. Dies wird in der Grafik durch einen SOC-Sollwert von 0 % dargestellt. Der Speicher wird so lange befüllt, bis die Temperaturen der Sensoren *TIRC323* und *TIRC322* des

Speichers das Austrittskriterium dieses Szenarios erfüllen. Anschließend ist der *Netzkollaps* wieder aktiv. Die Temperatur des Sensors *TIRC401* ist in Abbildung 6.11 dargestellt. Die häufigen Sollwertwechsel des Reglers *pid_MV301* und deren Auswirkungen auf das System sind im Verlauf deutlich anhand der Regelabweichungen bei Szenariowechsel zu erkennen. Neben den Änderungen des Sollwerts hat der Leistungsbezug von *load2* ebenfalls Auswirkungen auf den Temperaturverlauf. So ist bspw. der Knick in der Leistungsabnahme von *load2* zum Zeitpunkt 14:55 mit einer Verzögerung aufgrund der Rohrleitung zum Zeitpunkt t_3 in Abbildung 6.11 in der Regelabweichung von 3 °C sichtbar. Der *Netzkollaps* wird zum Zeitpunkt t_4 von der Leitwarte deaktiviert.

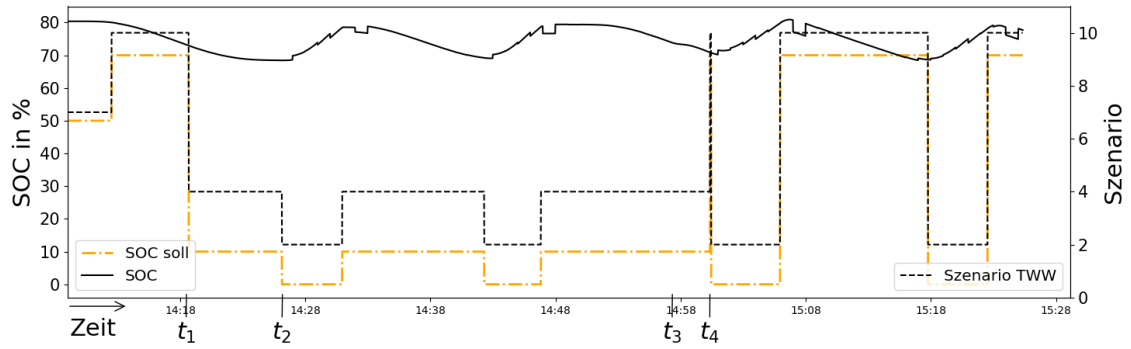


Abbildung 6.10.: Netzkollaps (iWÜST mit TWW, SOC)

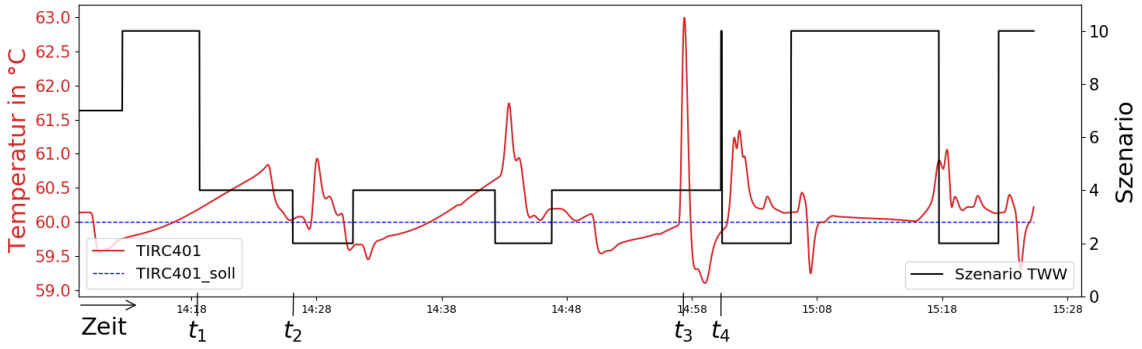


Abbildung 6.11.: Netzkollaps (iWÜST mit TWW, TIRC401)

Generell wird die Solltemperatur *TIRC401_soll* jedoch ausreichend genau geregelt. Die grafische Auswertung zeigt, dass die Anforderungen aus der Regelbeschreibung grundsätzlich erfüllt werden. Jedoch sind die daraus resultierenden Schwankungen in der Leistungsabnahme durch das häufige Be- und Entladen des Speichers für die Rückkehr in einen stabilen

Netzzustand nicht förderlich, weshalb eine andere Strategie für dieses Szenario entwickelt und getestet wurde.

Dabei wurde untersucht, ob durch die Veränderung der Prioritäten der Szenarien eine netzdienlichere Fahrweise ermöglicht werden kann. Die Solltemperatur am Sensor *TIRC401* konnte jedoch mit dieser Variante nicht gehalten werden. Die detaillierte Beschreibung dazu befindet sich im Anhang (Seite LXXXII). Des Weiteren wurde in dieser Simulation deutlich, dass bei einer Vorlauftemperatur von 70 °C der Speicher wegen des Sollwerts am Sensor *TIRC401* = 60 °C nur bis zu einem SOC-Wert von 65 % entladen werden kann, da dann das Szenario *Minimale Temperatur* ausgelöst wird. Zur Lösung dieses Problems könnte die Regelung des Aktors *MV301* so angepasst werden, dass ab einer bestimmten Grenztemperatur am Sensor *TIRC324* (bspw. T < 62 °C) die Versorgung der Last durch eine Mischung aus Netzvorlauf und Speicherinhalt erfolgt. Außerdem könnte in Betracht gezogen werden, dass eine alternative Berechnung des SOC verwendet wird. Bspw. könnte ein so genanntes *Totvolumen*, welches dem Bereich des Speichers entspricht, in dem die Temperatur unter der Solltemperatur *TIRC401_soll* liegt, berücksichtigt werden. Würden also nur die Temperaturen der oberen drei Sensoren über 60 °C liegen, so beträge der SOC 50 %.

Kommunikationsausfall

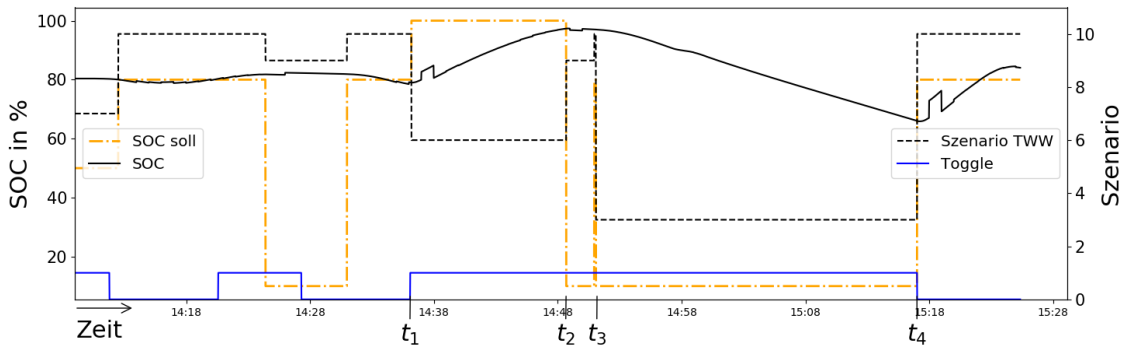


Abbildung 6.12.: Kommunikationsausfall (iWÜST mit TWW)[3]

In den vorherigen Simulationen wurde das Auslösen des *Kommunikationsausfalls* verhindert, indem mit einer Zählvariable der Wert des *Toggle*-Signals nach 7 bzw. 8 Minuten verändert wurde. Zur Validierung des Regelalgorithmus wurde das Verändern des *Toggle*-Signals in der Simulation zum Zeitpunkt t_1 ausgesetzt. Anhand Abbildung 6.12 ist ersichtlich, dass

[3] t_1 - t_2: *Speicher Überladen* aktiviert

die Anforderungen der Regelbeschreibung erfüllt werden. Der *Kommunikationsausfall* wird wie erwartet zum Zeitpunkt t_3 ausgelöst, da sich zu diesem Zeitpunkt das *Toggle*-Signal seit 15 Minuten nicht verändert hatte. Anschließend sinkt der SOC, bis sich der Wert des *Toggle*-Signals ändert (t_4) und kein *Kommunikationsausfall* mehr vorhanden ist.

Minimale Temperatur

Die Grenzwerte des Szenarios sind abhängig von der Netztemperatur (siehe Tabelle 6.4). Diese Abhängigkeit war in der Regelbeschreibung ursprünglich nicht berücksichtigt. Bei der Simulation wurde jedoch deutlich, dass diese Unterscheidung für eine geringe Netztemperatur benötigt wird. Die vorgegebenen Grenzwerte aus der Regelbeschreibung entsprechen den Werten, die bei einer Netztemperatur *T_Netz* größer als 75 °C verwendet werden. Sobald die Netztemperatur geringer als 70 °C ist, wird der Sollwert *TIRC324_soll_kopfraum* sowie der Grenzwert der Temperatur *TIRC324_ladung_ausstieg* abhängig von der aktuellen Netztemperatur berechnet.
Die Abbildung 6.13 zeigt, dass das gewünschte Regelverhalten aus der Regelbeschreibung erfüllt wird. Das Szenario wird zum Zeitpunkt t_2 ausgelöst, da die Temperaturen *TIRC324* und *TIRC323* unter die Grenzwerte (*TIRC324_min* und *TIRC32X_grenz*) sinken. Zum Zeitpunkt t_3 steigen die Temperaturen der Sensoren *TIRC324* und *TIRC322* über die Grenzwerte für das Austrittskriterium, sodass das Szenario deaktiviert wird.
Die Simulation wurde ebenfalls für eine Netztemperatur von 70 °C durchgeführt und ausgewertet. Die Ergebnisse können im Anhang (Seite A) eingesehen werden.

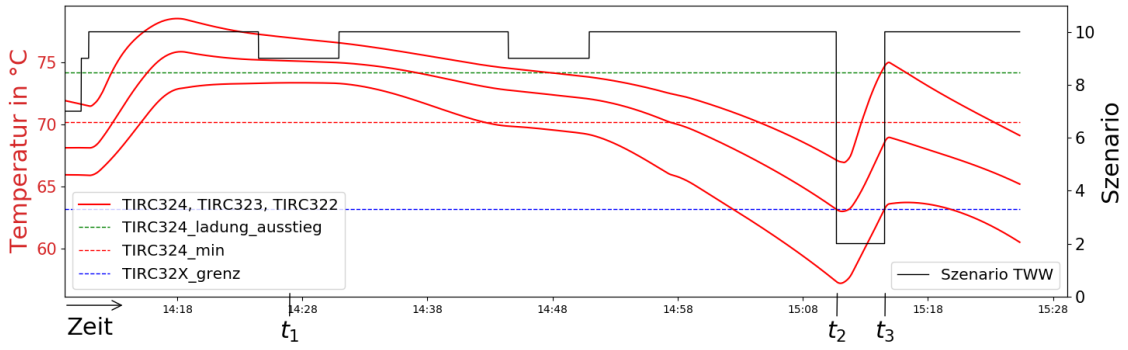


Abbildung 6.13.: Minimale Temperatur (iWÜST mit TWW)[4]

[4] t_1: SOC_soll = 40 %

Inbetriebnahme

Während der Inbetriebnahme wird die Nummer des aktuellen Tests (*Test-Nr.*) der Variable *Inbetriebnahme_Szenario_TWW* zugewiesen, welche von der Leitwarte ausgelesen werden kann. Im Folgenden werden die einzelnen Funktionstests beschrieben.

Test-Nr. 0: Ping-Test Der Ping-Test wird als bestanden angesehen, wenn die Variable *EXTERN_Inbetriebnahme_start_TWW* von der Leitwarte von *FALSE* auf *TRUE* gesetzt wurde. Durch dieses Signal wird die Inbetriebnahme gestartet.

Test-Nr. 1: Funktionstest Der Funktionstest wird gestartet, sobald die Variable *start_funktionspruefung_1_TWW* von dem Regelalgorithmus auf *TRUE* gesetzt wurde. Nachdem die Variable gesetzt wurde, wird die Funktionsprüfung aus Szenario 5 durchgeführt. Sobald der Ergebniscode der Variable *Ergebnis_funktionspruefung_TWW* den Wert 200 annimmt, gilt die Funktionsprüfung als abgeschlossen. Durch die erfolgreiche Durchführung der Funktionsprüfung ist neben den Ventilen *MV101* und *MV301* die Funktionalität des im Wärmemengenzähler integrierten Volumenstrommessers verifiziert.

Test-Nr. 2: Prüfung der Plausibilität der Sensoren Die Temperatursensoren des Systems werden verifiziert, indem für jeden Sensor überprüft wird, ob die Temperatur im Bereich von 0 °C-100 °C liegt.

Test-Nr. 3: Prüfung der STB-Prüftaste Durch Betätigen der STB-Prüftaste wird deren Funktionalität überprüft.

Test-Nr. 4: Verdrahtungstest Durch das manuelle Entfernen der steckbaren Temperaturfühler wird der Verdrahtungstest durchgeführt. Der Test war erfolgreich, wenn nach dem Trennen der Verbindung die Temperatur 850,5 °C ausgegeben wird. Dies entspricht der Temperatur, die bei einem maximalen Widerstandswert von der Klemme an die Entwicklungsumgebung übergeben wird.

Test-Nr. 5: Speicher laden *SOC_soll* = 80 %. Sobald der berechnete SOC größer als 78 % ist gilt der Test als erfolgreich abgeschlossen.

Test-Nr. 6: Speicher entladen *SOC_soll* = 20 % Sobald der berechnete SOC geringer als 22 % ist gilt der Test als erfolgreich abgeschlossen.

Sobald der letzte Test erfolgreich durchgeführt wurde, ist die Inbetriebnahme abgeschlossen. Die Validierung der *Inbetriebnahme* wird anhand der Simulationsergebnisse in Abbildung 6.14 durchgeführt. Die Variable *STB_pruef_TWW* wird zum Zeitpunkt t_2 auf *TRUE* gesetzt und simuliert so das Betätigen der *STB-Prüftaste*. Ein Verdrahtungsfehler in Form

einer Verbindungsunterbrechnung kann in der Simulationsumgebung nicht dargestellt werden. Aus diesem Grund können die Tests des *Verdrahtungstest* nicht erfolgreich abgeschlossen werden. Es ist ersichtlich, dass das Szenario nach dem Setzen der Variable *EXTERN_Inbetriebnahme_start_TWW* (t_1) korrekt gestartet wird. Anschließend werden die Tests 1, 2 und 3 erfolgreich durchgeführt. Wie erwartet kann der Verdrahtungstest nicht erfolgreich durchgeführt werden. Das Szenario ist damit im Bereich des Möglichen innerhalb der Simulationsumgebung hinreichend validiert. Die Hardwaretests konnten in der Hardwaretestumgebung erfolgreich durchgeführt werden.

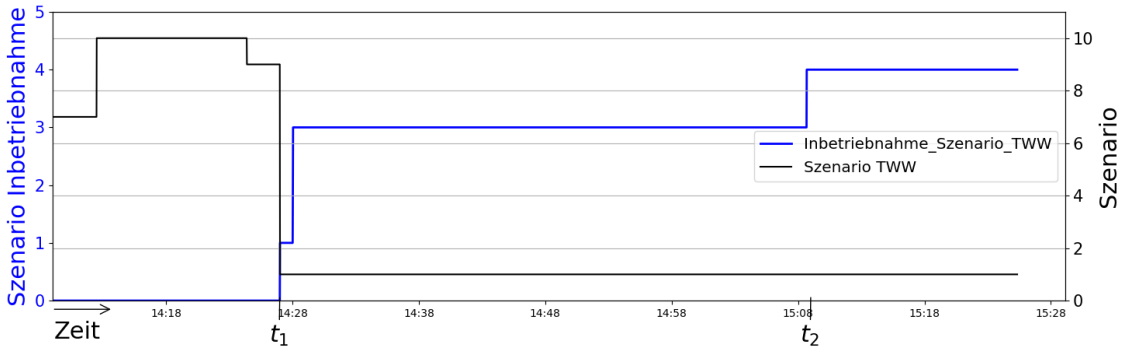


Abbildung 6.14.: Inbetriebnahme (iWÜST mit TWW)

Normalbetrieb (iWÜST ohne TWW)

Das Szenario *Normalbetrieb* ist das am niedrigsten priorisierte Szenario und dementsprechend automatisch aktiviert, wenn keines der anderen Szenarien ausgelöst wurde. Der Wert

Tabelle 6.8.: Veränderungen in der Parametrierung für die Validierung der jeweiligen Szenarien der iWÜST ohne TWW

Sz.-Nr.	Parameter	Zeitraum	Wert
8	*load1*	t_2 - t_3	55 kW
	load1	t_3 - t_4	5 kW
	load2	gesamter Simulationszeitraum	65 kW
6	*EXTERN_RLTemp_soll*	t_1 - t_2	40 °C
4	*TC101_Netzkollaps_soll*	gesamter Simulationszeitraum	60,15 °C
3	*TC101_vertrag_soll*	gesamter Simulationszeitraum	88 °C
2	*load1*	gesamter Simulationszeitraum	P_t(*load1*) - 10 kW
	dT_RLbegrenzung	gesamter Simulationszeitraum	10 °C

TC101_soll ist von verschiedenen Parametern abhängig. Der Programmablaufplan in Abbildung A.37 im Anhang visualisiert diese. Die grafische Auswertung der Reglerparametrierung des Reglers *pid_MV101* (Abbidung 5.11) in Kapitel 5.3 kann zur Validierung der Regelgüte herangezogen werden. Die Temperatur des Sensors *TC101* entspricht dabei dem Sollwert *TC101_soll*, sodass die Anforderungen der Regelbeschreibung erfüllt werden.

Rücklaufbegrenzung (iWÜST ohne TWW)

Es wurden drei Strategien zur *Rücklaufbegrenzung* untersucht. Dabei hat sich herausgestellt, dass die Varianten *"Kontinuierliche Rücklaufregelung auf einen Sollwert"* und *"Minimaler Öffnungswinkel des Ventils"* bzgl. der Anforderungen nicht geeignet sind. Bei einer konstant niedrigen Leistungsabnahme würde die Regelung ständig zwischen den Szenarien *Rücklaufbegrenzung* und *Normalbetrieb* pendeln. Dieses Pendeln kann lediglich durch eine starke Dämpfung bzw. einen Timer verlangsamt werden. Die Validierung dieser Strategien ist im Anhang auf Seite LXXXVIII beschrieben.
Der folgende Abschnitt befasst sich mit der Validierung der dritten Variante *"Regelung der Temperatur des Sensors TC101"*. Die Solltemperatur des Sensors *TC101* wird dabei bei einer zu hohen Rücklauftemperatur zunächst um $5\,^{\circ}\mathrm{C}$ vermindert. Im Anschluss wird die Solltemperatur *TC101_soll* im Abstand von 5 Minuten so lange um $3\,^{\circ}\mathrm{C}$ verringert, bis das Austrittskriterium erfüllt ist. Es wird neben der Temperatur *TIRC103* die Temperaturdifferenz der Sensoren *TIRC102* und *TC101* als Austrittskriterium verwendet. Dadurch soll die Leistungsabnahme durch die Last miteinbezogen werden. Bei einem konstanten Massenstrom (konstante Pumpenleistung der Pumpe *pump5*) ist die Temperaturdifferenz direkt proportional zur Leistungsabnahme (siehe Formel 2.3). Eine hohe Leistungsabnahme bedingt eine geringe Rücklauftemperatur, sodass dadurch eine Erhöhung der Last detektiert werden kann.
Für diese und die folgenden Simulationen wurde die Leistungsabnahme der Komponente *load2* durch eine konstante Last (30 kW) abgebildet. Dadurch ist die Rücklauftemperatur aus dem Wasserkreislauf *TWW* (*TIRC305*) relativ konstant, wodurch die Netztemperatur vom Regler *pid_generator* besser gehalten werden kann. Die grafische Auswertung der Ergebnisse wird dadurch erleichtert.
Die Simulationsergebnisse sind in den Abbildungen 6.15 und 6.16 dargestellt. Durch die treppenförmige Veränderung der Temperatur *TC101* sinkt die Rücklauftemperatur *TIRC103* (siehe Abbildung 6.15 ab t_1). Es sollte jedoch berücksichtigt werden, dass die Temperatur ebenfalls von der Last *load1* abhängig ist, welche zum Zeitpunkt 14:41 Uhr die geringste Abnahme hat. In Abbildung 6.16 ist der Verlauf der Temperaturdifferenz von *TC101* und *TIRC102* dargestellt. Der gezackte Verlauf ergibt sich durch die Rohrleitung zwischen den Temperatursensoren. Zunächst erfährt der Sensor *TC101* eine Temperaturänderung, welche am Sensor *TIRC102* erst erfasst wird, wenn das Wärmeträgermedium den Wasserkreislauf

Heizkreis durchflossen hat. Die *Rücklaufbegrenzung* wird deaktiviert, sobald die Temperatur unter den zulässigen Grenzwert *TIRC103_zulaessig* gesunken ist (t_2) und die Lastabnahme bei einem konstanten Massenstrom hoch genug ist (t_3). Der Regelalgorithmus erfüllt somit die Anforderungen aus der Regelbeschreibung.
Für eine Verwendung sollte der Wert *dT_RLbegrenzung*, welcher für diese Überprüfung pauschal mit 20 °C angenommen wurde, untersucht und eventuell hinsichtlich der Last optimiert werden. Die Berücksichtigung der Last funktioniert wie in diesem Fall nur bei einem konstanten Massenstrom im Wasserkreislauf *Heizkreis*. Bei einem inkonstanten Massenstrom könnte in Zukunft die Veränderung des Steuersignals der Pumpe *pump5* berücksichtigt werden.

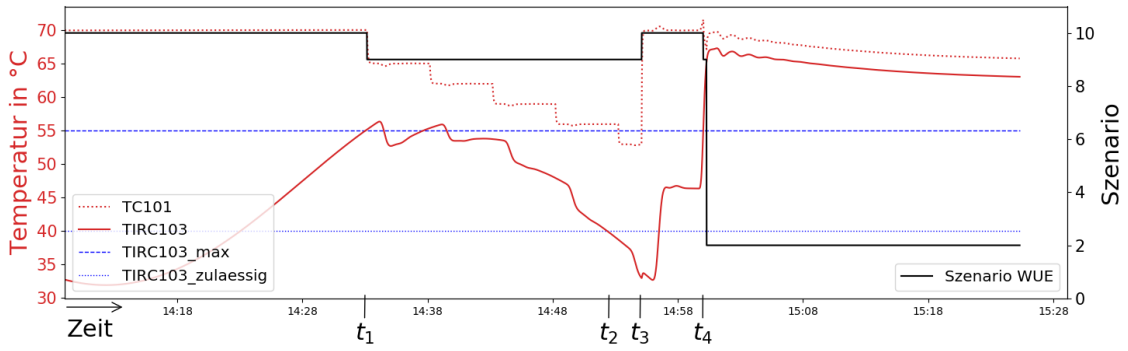


Abbildung 6.15.: Rücklaufbegrenzung (iWÜST ohne TWW, TC101, TIRC103)[5]

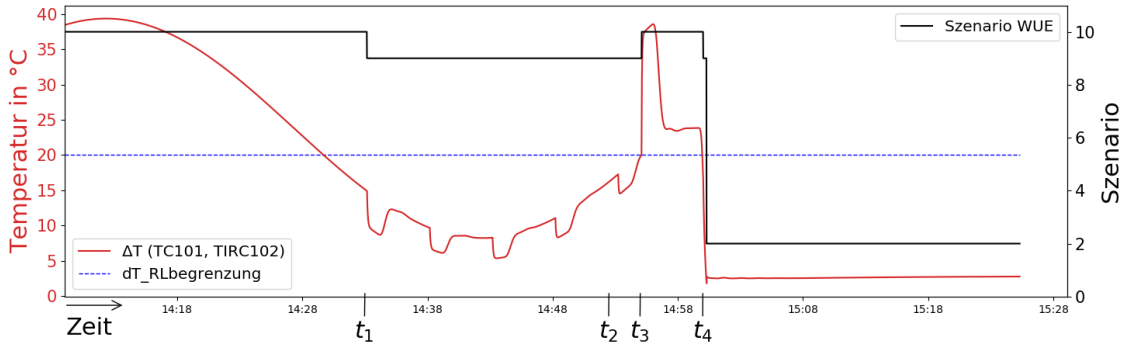


Abbildung 6.16.: Rücklaufbegrenzung (iWÜST ohne TWW, Temperaturdifferenz TC101 und TIRC103)[5]

[5] t_4: *Rücklaufbegrenzung* kurz aktiviert, anschließend *Warmhaltung*

Leistungsbegrenzung (iWÜST ohne TWW)

Bei einer ersten Simulation mit konstanter Lastabnahme über dem Grenzwert wurde ersichtlich, dass die Auswirkungen der *Leistungsbegrenzung* mit dem verwendeten Modell nicht gezeigt werden können. Die Ursache dafür liegt in dem Modell der Last *load1*. Die Verringerung des Massenstroms durch den Aktor *MV101* hat zur Folge, dass die Austrittstemperatur der Last verringert wird, da sich die Austrittstemperatur nach Formel 2.3 proportional zum Massenstrom verhält. Die Komponente *load1* bezieht, unabhängig von der Umgebungstemperatur, immer die vorgegebene Leistung, sodass die Austrittstemperatur sogar unter 0 °C fällt. Die Leistungsabnahme der Last wirkt sich also nur auf die Austrittstemperatur aus, weshalb die Validierung mit diesem Modell nur bedingt durchführbar ist. Auf Seite XC im Anhang wird darauf näher eingegangen.

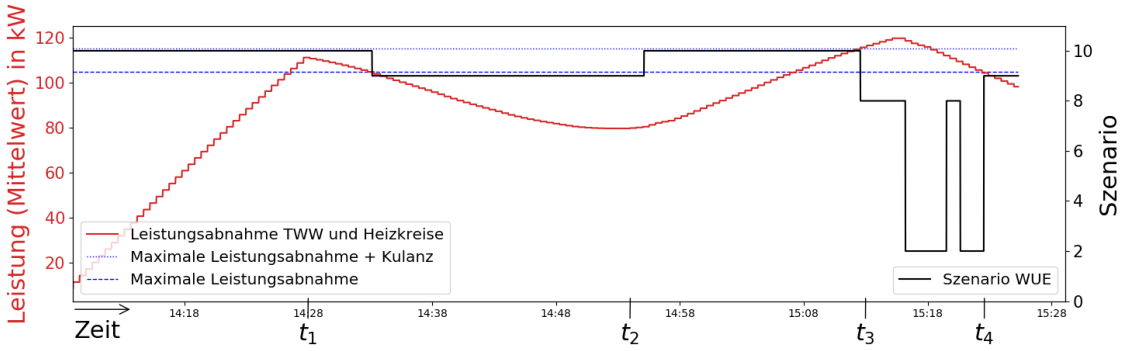


Abbildung 6.17.: Leistungsbegrenzung (iWÜST ohne TWW)[6]

Die Leistungswerte der Lasten *load1* und *load2* wurden so angepasst (Tabelle 6.8), dass das Eintritts- und Austrittskriterium im betrachteten Zeitraum erfüllt ist und somit zumindest deren Funktionaltiät gezeigt werden kann. Außerdem wurde die *Leistungsbegrenzung* der iWÜST mit TWW deaktiviert.
Abbildung 6.17 zeigt, dass die *Leistungsbegrenzung* nach Überschreiten des oberen Grenzwerts aktiviert wird (t_3). Zum Zeitpunkt t_4 sinkt der Mittelwert der abgenommenen Leistung unter den Grenzwert des Austrittskriterium und die niedriger priorisierte *Rücklaufbegrenzung* ist aktiviert. Dies entspricht den Anforderungen der Regelbeschreibung. Der Mittelwert der Leistung steigt bis zum Zeitpunkt t_1 an, da zuvor Leistungswerte, die zur Bestimmung des Mittelwerts verwendet werden, noch den Initalwert 0 kW betragen.
Neben der Begrenzung der Leistung durch die Verringerung der sekundären Vorlauftemperatur *TC101* wird in der Regelbeschreibung vorgeschlagen, dass der Aktor *MV101* verwendet

[6] t_1 - t_2: Leistungsabnahme entspricht Kurve der Lastabnahme von *load1*

wird, um auf einen Leistungswert zu regeln, der geringer als die vertraglich vereinbarte maximale Leistung (*P_WMZ001_zulaessig*) ist. Diese Variante konnte im Umfang dieser Arbeit aufgrund der Modellvereinfachungen nicht durchgeführt werden. Durch die Verwendung der Ventilstellungen zur Berechnung der abgenommenen Leistung würde die Regelgröße der Stellgröße entsprechen. Diese Variante kann erst getestet werden, sobald ein Modell des Wärmemengenzählers erstellt wurde.

Rücklauftemperaturregelung (iWÜST ohne TWW)

Die Regelbeschreibung sieht vor, dass durch die Aktivierung der *Rücklauftemperaturregelung* von der Leitwarte eine definierte Rücklauftemperatur eingestellt wird. Dafür werden Regelparameter für eine Rücklauftemperaturregelung (Kp = $2 \cdot 10^{-6}$, Tn = 500, Tv = 100) verwendet, die durch die *empische Methode zur Reglereinstellung* ermittelt wurden. Die Regelung besitzt eine hohe Dämpfung, damit kein Resonanzverhalten zwischen den Reglern *pid_generator* und *pid_MV101* eintreten kann. Die hohe Dämpfung zeigt sich in dem trägen Regelverhalten. Der Fokus dieser Untersuchung liegt darauf, dass bei Aktivierung von Szenario 6 die Rücklauftemperaturregelung generell aktiviert wird. Ein exaktes Regeln auf den Sollwert ist dafür nicht notwendig.
Die Validierung der *Rücklauftemperaturregelung* erfolgt anhand Abbildung 6.18. Das Szenario wird im Zeitraum t_1 - t_2 von der Leitwarte ausgelöst. Der Sollwert wird nach dem Einschwingverhalten ausreichend genau erreicht.

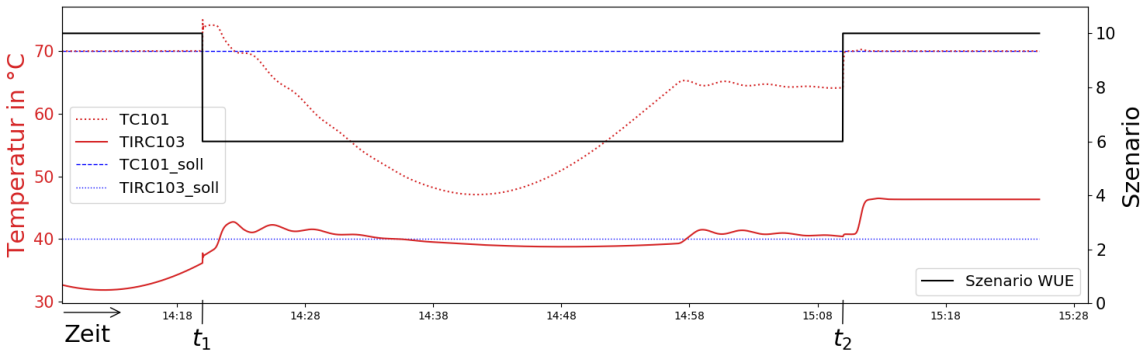


Abbildung 6.18.: Rücklauftemperaturregelung (iWÜST ohne TWW)[7]

[7] t_1 - t_2: *Rücklauftemperaturregelung* aktiviert, Sollwert der Regelung nicht *TC101*

Funktionsprüfung (iWÜST ohne TWW)

Im Abschnitt *Funktionsprüfung* (iWÜST mit TWW) ist die Validierung dieses Szeanrios enthalten. Die Simulationsergebnisse zeigen, dass die Anforderungen aus der Regelbeschreibung erfüllt werden.

Netzkollaps (iWÜST ohne TWW)

Der *Netzkollaps* kann anhand Abbildung 6.19 validiert werden. Sobald das Szenario von der Leitwarte aktiviert wird (t_1), stellt sich eine neue Temperatur am Sensor *TC101* aufgrund des veränderten Sollwerts *TC101* ein. Ab dem Zeitpunkt der Deaktivierung (t_2) entspricht die Temperatur am Sensor *TC101* dem Sollwert des *Normalbetriebs*. Dieses Verhalten entspricht den Anforderungen aus der Regelbeschreibung.

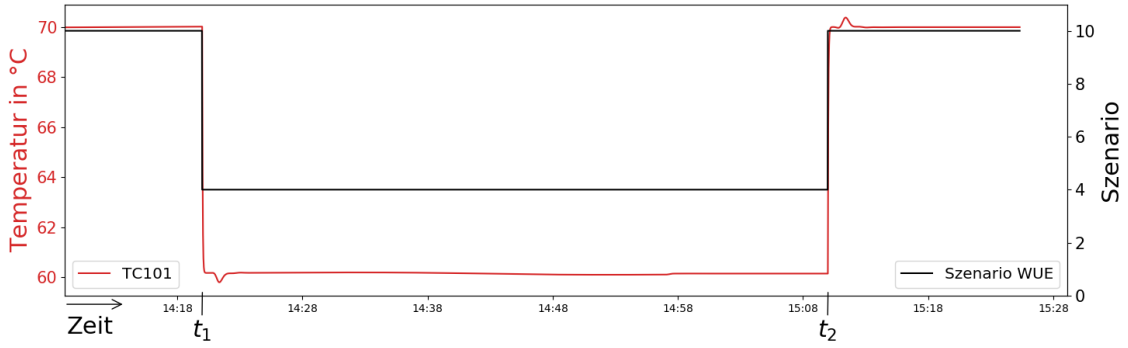


Abbildung 6.19.: Netzkollaps (iWÜST ohne TWW)

Kommunikationsausfall (iWÜST ohne TWW)

Die Aktivierung und Deaktivierung des *Kommunikationsausfalls* erfolgt durch das Flag *FLAG_3*, welches vom Regelalgorithmus der iWÜST mit TWW bei einem Kommunikationsausfall auf *TRUE* gesetzt wird.
Der Sollwert *TC101_soll* beträgt in der Simulation zu diesem Szenario 88 °C. Dadurch kann die Funktionalität der Berechnung des neuen Sollwerts mit Hilfe der Grädigkeit gezeigt werden. Der hohe Sollwert hat zur Folge, dass die *Rücklaufbegrenzung* wegen der hohen Rücklauftemperatur aktiviert ist, bis das Szenario *Kommunikationsausfall* zum Zeitpunkt t_2 eintritt. Anschließend beträgt der Sollwert *TC101_soll* = *T_Netz_mittel* - *T_WUT_graedigkeit* = 85 °C (siehe Abbildung 6.20). Die Anforderungen werden somit vom Regelalgorithmus erfüllt.

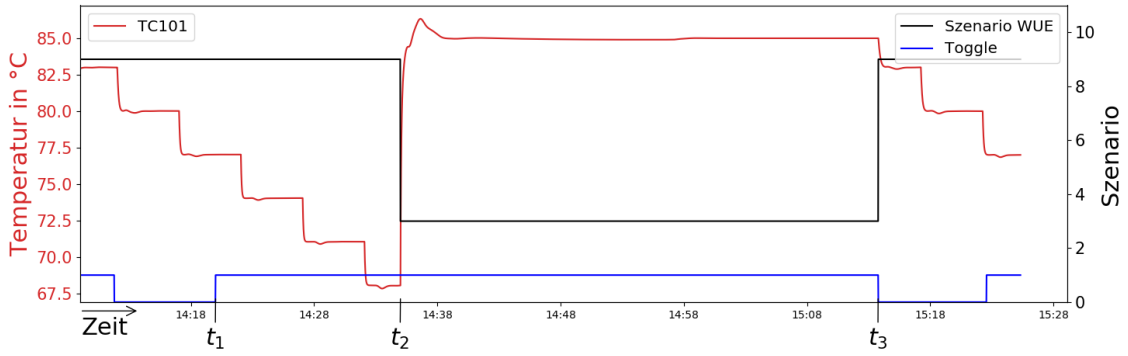


Abbildung 6.20.: Kommunikationsausfall (iWÜST ohne TWW)[8]

Warmhaltung (iWÜST ohne TWW)

Die Lastkurve *load1* wurde für die Validierung der *Warmhaltung* angepasst. Die abgenommene Leistung wurde in jedem Zeitschritt im Gegensatz zum Wert aus der Lastkurve des *Testsimulations* um 10 kW reduziert (siehe Abbildung A.43 im Anhang). Durch die Verwendung der veränderten Lastkurve wurde deutlich, dass der in Kapitel 6.6.2 verwendete Grenzwert *dT_RLbegrenzung* = 20 °C für diese Parametrierung zu groß ist, da die *Rücklaufbegrenzung* nach dem Auslösen nicht mehr deaktiviert wurde. Dieser Grenzwert *dT_RLbegrenzung* beträgt deswegen für diese Untersuchung 10 °C.
In Abbildung 6.21 ist auf der linken y-Achse der Ventilöffnungswinkel in % aufgetragen. Das Ausgangssignal *MV101_* des Reglers *pid_MV101* wurde in einen fiktiven Öffnungwinkel umgerechnet und ist in der Grafik als *Ausgangssignal pid_MV101* gekennzeichnet. Das Stellsignal *MV101* an die Pumpe *MV101* wurde in einen Öffnungswinkel umgerechnet und ist in der Grafik als durchgezogene, grüne Linie eingezeichnet. Die *Warmhaltung* wird zum Zeitpunkt t_1 korrekt ausgelöst, da ab diesem Zeitpunkt der Wert der Variable *Ausgangssignal pid_MV101* kleiner als der Grenzwert *MV101_offen_min* ist. Der Öffnungswinkel beträgt anschließend so lange 5 %, bis die Lastabnahme von *load1* so groß ist, dass die Solltemperatur am Sensor *TC101* nur durch einen größeren Öffnungwinkel gehalten werden kann (t_2). Die Temperatur *TC101* liegt im betrachteten Zeitraum über dem Sollwert *TC101_soll*, weshalb das Stellsignal *MV101_* des Reglers *pid_MV101* einen fiktiven Öffnungswinkel von 0 % einstellt. Das Regelverhalten des Szenarios entspricht somit dem, was in der Regelbeschreibung gefordert wird.

[8] t_1: Toggle-Signal deaktiviert, t_3: Toggle-Signal aktiviert

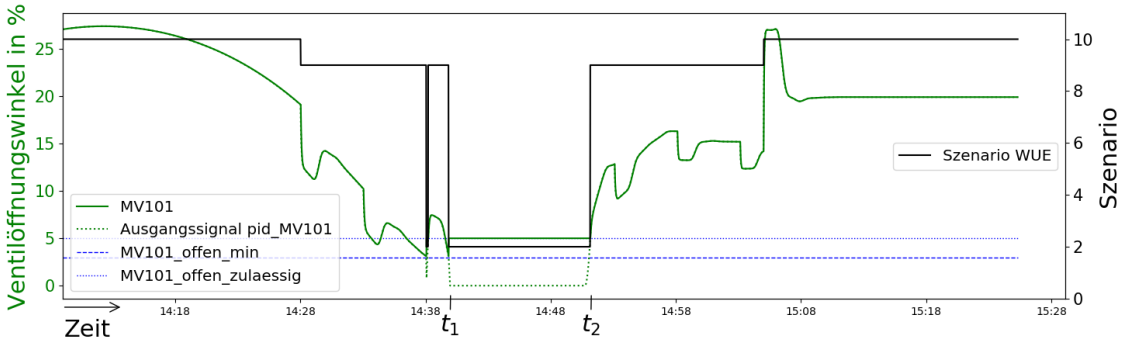


Abbildung 6.21.: Warmhaltung (iWÜST ohne TWW)

Inbetriebnahme (iWÜST ohne Tww)

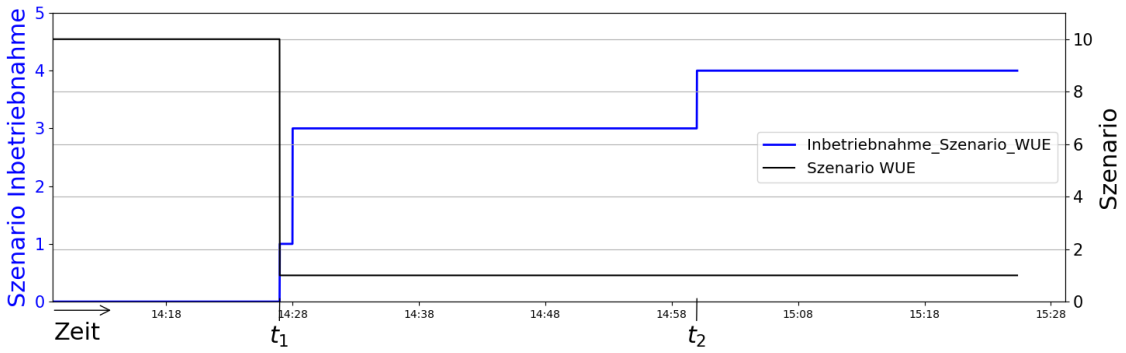


Abbildung 6.22.: Inbetriebnahme (iWÜST ohne TWW)[9]

Der Regelalgorithmus der *Inbetriebnahme* für die iWÜST ohne TWW unterscheidet sich zu dem der iWÜST mit TWW nur insofern, als dass keine Prüfung des Speichers durchgeführt wird (kein Test-Nr. 5 und 6). Ansonsten ist der Ablauf gleich und kann der Beschreibung für die iWÜST mit TWW entnommen werden. Die Validierung der *Inbetriebnahme* der iWÜST ohne TWW erfolgt anhand Abbildung 6.22. Die einzelnen Prüfschritte sind durch die Kennlinie *Inbetriebnahme_Szenario_WUE* dargestellt. Die Plausibilitätsprüfung der Sensoren (*Test-Nr. 2*) ist in der Grafik nicht dargestellt, da der Test sofort bestanden wird und im gleichen Zeitschritt mit dem Test *Prüfung der STB-Prüftaste* fortgefahren wird. Der Verdrahtungstest kann in der Simulationsumgebung nicht durchgeführt werden, weshalb dieser Test nicht erfolgreich abgeschlossen wird und das Szenario 4 der Inbetriebnahme bis zum Simulationsende aktiviert ist. Die Funktionalität des Regelalgorithmus wurde damit unter Berücksichtigung der

[9] t_1: Aktivierung *Inbetriebnahme* durch Leitwarte, t_2:*STB_pruef_WUE = TRUE*

modellbasierten Einschränkungen gezeigt. Wie bei der iWÜST mit TWW konnte Test 4 in der Hardwaretestumgebung erfolgreich durchgeführt werden.

Überprüfung des Überdruck-/Übertemperaturbegrenzer (iWÜST ohne TWW)

Dieses Szenario ist nicht in der Regelbeschreibung vorgesehen, wurde jedoch eingeführt, damit dieser Systemzustand von der Leitwarte detektiert werden kann. In Abbildung 6.23 entspricht der y-Achsenwert 10 des Szenarios dem bool'schen Wert *FALSE* und 11 dem bool'schen Wert *TRUE*. Die Begrenzer wurden in der Simulationsumgebung zu den Zeitpunkten t_1 (*TZ*) und t_2 (*TS*) für 5 Minuten aktiviert. In diesem Bereich beträgt der Massenstrom $0\,\mathrm{kg/s}$. Dies entspricht den Forderungen der Regelbeschreibung.

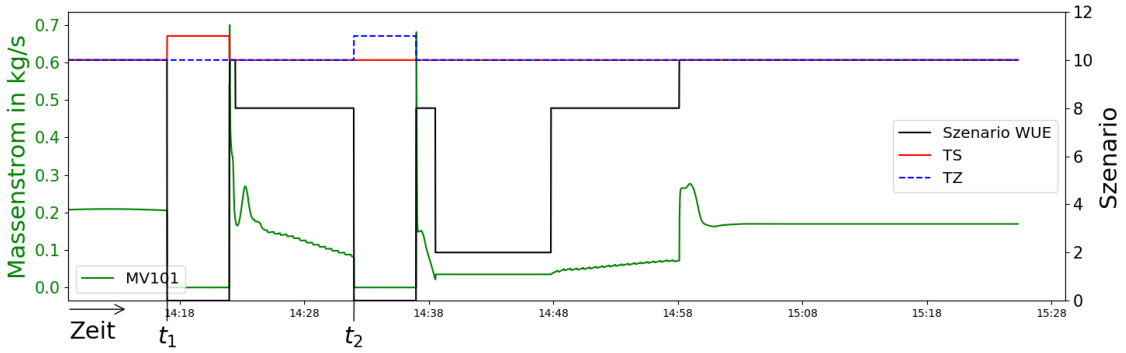


Abbildung 6.23.: Überdruck-/Übertemperaturbegrenzung (iWÜST ohne TWW)

Regelung des Aktors *MV302*

Die Regelung des Aktors *MV302* in das Anwenderprogramm *SETOUTPUT()* implementiert. Die Funktionsweise des Codes wird im Abschnitt *Regelung der Pumpe MV302* des Kapitels 5.3 beschrieben und validiert.

6.7. Ergebnis

Grundsätzlich kann an dieser Stelle festgehalten werden, dass das allgemeine Vorgehen zur Validierung des Regelalgorithmus und die damit einhergehende Überprüfung des Hard- und Softwaremodells sowie der Programme aus dem Kapitel 3 (Parser und CODESYSImporter)

erfolgreich war.
Die Hardwaretestumgebung erfüllt die Ansprüche bzgl. Qualität und Funktionalität. Lediglich die verwendeten Bananenbuchsen der Schalttafel entsprechen nicht den Güteanforderungen. Durch das Einstecken und Entfernen der Stecker lösten sich einige der Buchsen. Sie sollten für eine weitere Verwendung durch Buchsen eines anderen Fabrikats ausgetauscht werden.
Die Validierung der Szenarien offenbarte zwei Schwächen des Simulationsmodells. Die Trägheit eines realen Wärmenetzes wird im Modell unzureichend abgebildet, sodass dadurch regelungstechnische Probleme auftraten. Außerdem kann durch den verwendeten Berechnungsansatz des SOC in der Simulationsumgebung nicht die komplette Kapazität des Speichers verwendet werden. Dabei ist allerdings zu beachten, dass sich die Simulationsumgebung zum Zeitpunkt der Nutzung für diese Arbeit noch in der fortlaufenden Entwicklung befand und somit auf keine Erfahrungswerte bzgl. Parametrierung und SOC-Berechnung zurückgegriffen werden konnte.
Das Importieren und Übersetzen des Steuerungsprogramms (Export-Datei) von *CoDeSys 2.3* in *Jarvis* mittels der Programme CODESYSImporter und STParser aus Kapitel 3 funktionierte fehlerfrei.
Die Szenarien ließen sich in beiden Testumgebungen auslösen, anpassen und bzgl. ihrer Anforderungen validieren. In der folgenden Tabelle 6.9 sind die daraus resultierenden Ergebnisse dargestellt. Die einzelnen Szenarien lassen sich an Hand von drei Kriterien bewerten. Es wird neben dem primären Ziel der Validierung (Erfüllen der **Anforderungen** aus der Regelbeschreibung) die **Übertragbarkeit** der Validierungsergebnisse auf ein reales System betrachtet. Außerdem wird bewertet, inwiefern ein Szenario innerhalb der Hardwaretestumgebung (teilweise mit der Entwicklungsumgebung *CoDeSys 2.3*) ausgelöst und das Regelverhalten dadurch veranschaulicht werden kann (**Visualisierbarkeit**). Die Skala erstreckt sich dabei von *gut* (+) über *befriedigend* (o) bis *schlecht* (-). Wenn keine Bewertung abgegeben werden kann wird dies durch das Zeichen (x) gekennzeichnet.
Anhand der Bewertungen aus Tabelle 6.9 ist ersichtlich, dass ein Großteil der Szenarien die Anforderungen der Regelbeschreibung erfüllt. Diese Ergebnisse lassen sich in der Regel auf die reale Anlage übertragen und können anhand der Hardwaretestumgebung visualisiert werden. Aufgrund des fehlenden *Wärmemengenzählers* in beiden Modellen kann abschließend keine endgültige Aussage darüber getroffen werden, ob die Validierungsergebnisse der Szenarien *Funktionsprüfung* und *Inbetriebnahme* auf eine reale Anlage übertragbar sind.
Die Ergebnisse der *Füllstandsbegrenzung* können in der Hardwaretestumgebung aufgrund der statischen Temperaturen nicht veranschaulicht werden. Wegen der geringen Trägheit des simulierten Wärmenetzes kann anhand der durchgeführten Simulation nicht eruiert werden, ob die Ergebnisse des Szenario *Rücklaufbegrenzung* der iWÜST ohne TWW für die reale Anlage ebenfalls gelten. Die Validierungserkenntnisse der *Leistungsbegrenzung* (iWÜST ohne TWW) können wegen des Modells *load1* nicht gänzlich auf eine reale Anlage übertragen werden, da in dem Modell die Umgebungstemperatur nicht berücksichtigt wird.

Tabelle 6.9.: Auswertung der Validierung der Regelalgorithmus

Szenario	**Anforderungen**	**Übertragbarkeit**	**Visualisierbarkeit**
iWÜST mit TWW			
Normalbetrieb	+	+	+
Rücklaufbegrenzung	+	+	+
Leistungsbegrenzung	+	+	+
Füllstandsbegrenzung	-	+	-
Speicher Überladen	+	+	+
Funktionsprüfung	+	x	+
Netzkollaps	+	+	+
Kommunikationsausfall	+	+	+
Minimale Temperatur	+	+	+
Inbetriebnahme	+	x	+
iWÜST ohne TWW			
Normalbetrieb	+	+	+
Rücklaufbegrenzung	o	x	+
Leistungsbegrenzung	o	-	o
Rücklauftemperaturregelung	+	o	+
Funktionsprüfung	+	x	+
Netzkollaps	+	+	+
Kommunikationsausfall	+	+	+
Warmhaltung	+	+	+
Inbetriebnahme	+	x	+
Überdruck-/Übertemperatur	+	+	+

7. Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde der Regelalgorithmus einer intelligenten Wärmeübergabestation (iWÜST) entwickelt und mittels einer Hardwaretestumgebung sowie einem Softwaremodell validiert. Dafür wurde neben den Modellen der iWÜST ein Tool erstellt, welches die Verwendung von Steuerungscodes der Entwicklungsumgebung *CoDeSys 2.3* in der Simulationsumgebung *Jarvis* ermöglicht.
Das Tool umfasst hauptsächlich zwei Programme zum Importieren und Übersetzen von Regelalgorithmen. Es wurde in der Programmiersprache Python realisiert und in die Simulationsumgebung *Jarvis* integriert. Das Programm *CODESYSImporter* extrahiert dabei zunächst alle relevanten Informationen aus dem Steuerungsprogramm des Dateiformats ".exp". Der Parser (*STParser*) übersetzt anschließend die Anwenderprogramme der Programmiersprache "Strukturierter Text" in die Sprache Python. Unter Berücksichtigung weniger Nomenklaturen und Beschränkungen hinsichtlich verwendbarer Funktionen kann dadurch die Regelung einer SPS aus der Entwicklungsumgebung *CoDeSys 2.3* importiert und in der Simulationsumgebung *Jarvis* verwendet werden.
Der Fokus der SPS-basierten Hardwaretestumgebung lag auf einer möglichst realitätsnahen Nachbildung der steuerungstechnischen Komponenten. Dies wurde mittels eines Schaltkastens und einer Schalttafel umgesetzt. Der Schaltkasten enthält eine SPS und mehrere Klemmanschlüsse. Auf der Schalttafel sind alle Sensoren und Aktoren der iWÜST in Form von verstellbaren Potentiometern und Schaltern sowie Amperemetern und einer LED nachgebildet.
Das Softwaremodell der iWÜST wurde in der Simulationsumgebung *Jarvis* erstellt und ermöglicht eine detaillierte Analyse der Systemzustände bei unterschiedlichen Regelvorgängen. Die druckbehafteten Ventile der realen Anlage wurden im Modell durch Pumpen ersetzt, da der damalige Entwicklungsstand der Simulationsumgebung eine druckbehaftete Systemberechnung noch nicht zuließ.
Der Regelalgorithmus zur Steuerung der iWÜST ist in der Entwicklungsumgebung *CoDeSys 2.3* erstellt und anschließend anhand der beiden iWÜST-Modelle validiert worden. Dafür wurde der Steuerungscode in der Hardwaretestumgebung getestet, mittels des Tools zum Importieren und Übersetzen in die Simulationsumgebung geladen und anschließend durch die grafische Analyse verschiedener Testsimulationen ausgewertet. Mit diesem Vorgehen konnte neben dem Regelalgorithmus auch die Funktionsweise der Programme *CODESYSImporter* und *STParser* überprüft werden. Die Validierung offenbarte aufschlussreiche

Erkenntnisse hinsichtlich der Szenarien und der Berechnungsmethode des SOC, welche für die Weiterentwicklung der iWÜST hilfreich sind. So konnte grundsätzlich festgestellt werden, dass das gewünschte Systemverhalten durch die verschiedenen Regelszenarien in den meisten Fällen ausgelöst wird. Lediglich der Regelalgorithmus zur *Füllstandsbegrenzung* des Speichers hatte nicht die erwarteten Auswirkungen auf das System. Ferner wurde deutlich, dass eine abschließende Beurteilung einiger Szenarien aufgrund von Modellannahmen hinsichtlich der Verwendbarkeit in einer realen Anlage nicht ohne Vorbehalte möglich ist.

Zum Zeitpunkt der Fertigstellung dieser Arbeit war der Entwicklungsstand der Simulationsumgebung *Jarvis* parallel so weit fortgeschritten, dass eine Verwendung der druckbehafteten Simulation mit Ventilen möglich ist. Somit sollte im Anschluss dieser Arbeit ein druckbehaftetes Modell der iWÜST erstellt werden.
Der mittlerweile als Komponente erstellte *Wärmemengenzähler* kann in diesem Modell verwendet werden. Dies erfordert eine Anpassung des Regelalgorithmus an die dadurch erfolgten, veränderten Modelleigenschaften. Dabei soll zusätzlich eine andere Berechnungsmethode des SOC und eine hinsichtlich der Validierungsergebnisse angepasste Regelbeschreibung implementiert und getestet werden.
Für eine weitere Benutzung der Hardwaretestumgebung ist ein Austausch der Bananenbuchsen erforderlich.
Des Weiteren kann zukünftig in Betracht gezogen werden, die Funktionalität des Tools zum Importieren und Übersetzen der Steuerungsprogramme in die Simulationsumgebung zu erweitern. Funktionen, die in der aktuellen Version des Tools nicht berücksichtigt wurden, können hinzugefügt werden. Zusätzlich kann untersucht werden, ob die Verwendung anderer IEC61131-3-Programmiersprachen der Entwicklungsumgebung *CoDeSys 2.3* durch eine Erweiterung des Tools möglich und umsetzbar ist.

Literaturverzeichnis

[1] Zeit Online. *Ein neuer Weltklimavertrag*. URL:**http://www.zeit.de/thema/klimagipfel-2015**. Eingesehen am 03.05.2019.

[2] Bundesministerium für Wirtschaft und Energie. *Einsetzung der Kommission Wachstum, Strukturwandel und Beschäftigung*. URL:**https://www.bmwi.de/Redaktion/DE/Downloads/E/einsetzung-der-kommission-wachstum-strukturwandel-beschaeftigung.pdf?__blob=publicationFile**. Eingesehen am 03.05.2018.

[3] Richard Zahoransky. *Energietechnik; Systeme zur Energieumwandlung.Kompaktwissen für Studium und Beruf*. Springer Vieweg, 2014.

[4] RWTH Aachen HAW Hamburg, HE. *Smart Power Hamburg*. URL:**http://www.smartpowerhamburg.de**. Eingesehen am 03.05.2019.

[5] HAMBURG ENERGIE. *Smart Heat Grid Hamburg*. URL: **https://www.hamburgenergie.de/ueber-uns/unternehmen/forschungsprojekte/smart-heat-grid-hamburg/**. Eingesehen am 03.05.2019.

[6] IBA Hamburg. *Energiebunker*. URL:**https://www.iba-hamburg.de/projekte/energiebunker/projekt/energiebunker.html**. Eingesehen am 03.05.2019.

[7] Andreas Schneller, Leonard Frank, and Walter Kahlenborn. *Wärmenetze 4.0 im Kontext der Wärmewende - Politische Handlungsempfehlungen für eine Dekarbonisierung der leitungsgebundenen Wärmeversorgung*, 2018.

[8] Peter Lorenzen. Das Wärmenetz als Speicher im Smart Grid: Betriebsführung eines Wärmenetzes in Kombination mit einem stromgeführten Heizkraftwerk. Masterthesis, Hochschule für Angewandte Wissenschaften Hamburg, 2013.

[9] Kälte und KWK e.V. AGFW Der Energieeffizienzverband für Wärme. *AGFW - Hauptbericht 2016*, 2016.

[10] Jan Eric Thorsen, Henrik Lund, and Brian Vad Mathiesen. *Progression of District Heating - 1st to 4th generation*, 2018.

[11] Hamburg Energie. *Internes Dokument*. nicht veröffentlicht, 2018.

[12] VDI e.V. *VDI-Wärmeatlas*. Springer Vieweg, 2013.

[13] Arbeitsgemeinschaft QM Fernwärme. *Planungshandbuch Fernwärme, Schlussbericht*, 2017.

[14] Volker Quaschning. *Sektorkopplung durch die Energiewende*. Hochschule für Technik und Wirtschaft Berlin, 2016.

[15] Philipp Janßen. Smart Pro HeaT, (Powerpoint-Präsentation). 2018.

[16] F. Ebel, S.IDLER, G.Prede, and D. Scholz, editors. *Grundlagen der Automatisierungstechnik*. Festo Didactic GmbH und Co. KG, 2008.

[17] WAGO Kontakttechnik GmbH und Co. KG. *WAGO-I/O-SYSTEM 750, Programmierbarer Feldbuscontroller ETHERNET*. URL:**http://global.wago.com/media/2_products/m07500880_00000000_0de.pdf**. Eingesehen am 03.05.2019.

[18] WAGO Kontakttechnik GmbH & Co.KG. *WAGO-I/O-SYSTEM 750/753*. URL:**https://www.wago.com/de/automatisierungstechnik/io-systeme-entecken/750**. Eingesehen am 03.05.2019.

[19] Prof. Dr.-Ing. Ulfert Meiners. *Embedded Control - Vorlesungsskript*. 2017.

[20] R. Bliesener, F. Ebel, and C. Löffler, editors. *Speicherprogrammierbare Steuerungen - Grundstufe*. FESTO DIDACTIC KG, 1997.

[21] 3S Smart Software Solutions GmbH, editor. *Handbuch für SPS-Programmierung mit CoDeSys 2.3*. 3S - Smart Software Solutions GmbH, 2010.

[22] Paul McGuire. *Module pyparsing*. URL:**https://pythonhosted.org/pyparsing/pyparsing-module.html**. Eingesehen am 03.05.2019.

[23] Jan-Philip Beck. Modellierung und Simulation des Energiebunkers Wilhelmsburg als Grundlage einer Konzeptentwicklung zur Betriebsoptimierung und Netzrückspeisung. Masterthesis, Hochschule für Angewandte Wissenschaften Hamburg, 20138.

[24] Manfred Schleicher. *Regelungstechnik - Grundlagen und Tipps für den Praktiker*. JUMO GmbH & Co. KG, 2014.

A. Allgemeine Ergänzungen

Regelbeschreibungen

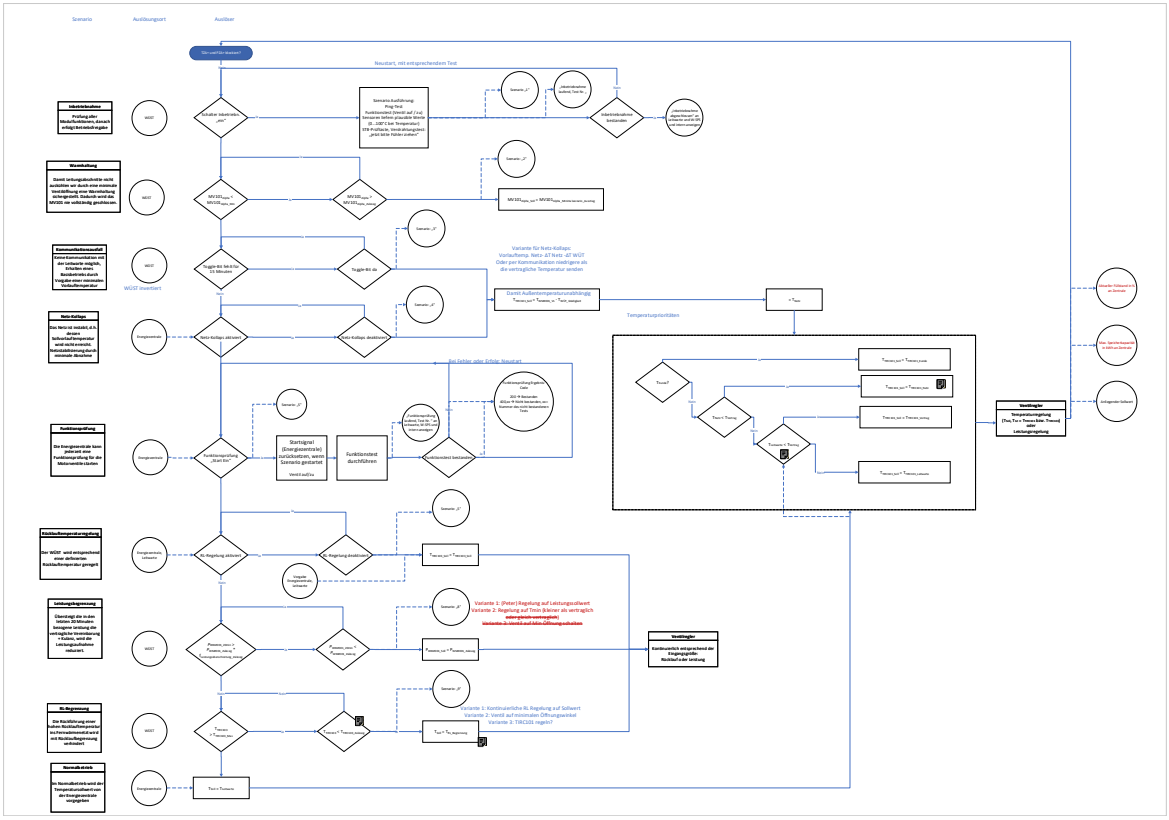


Abbildung A.1.: Regelbeschreibung iWÜST ohne TWW

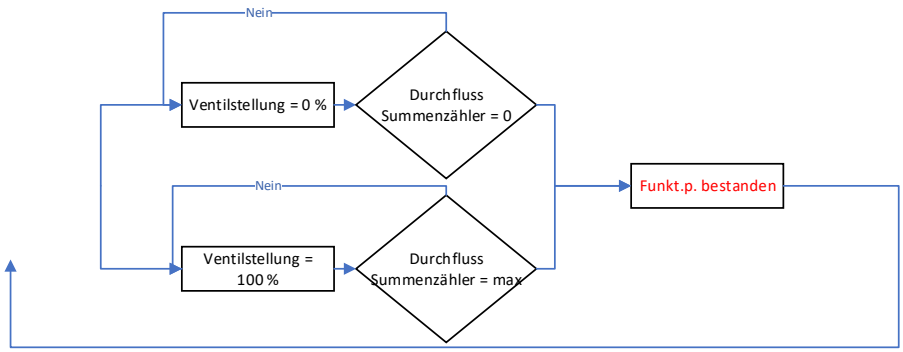


Abbildung A.2.: Regelbeschreibung der Funktionsprüfung

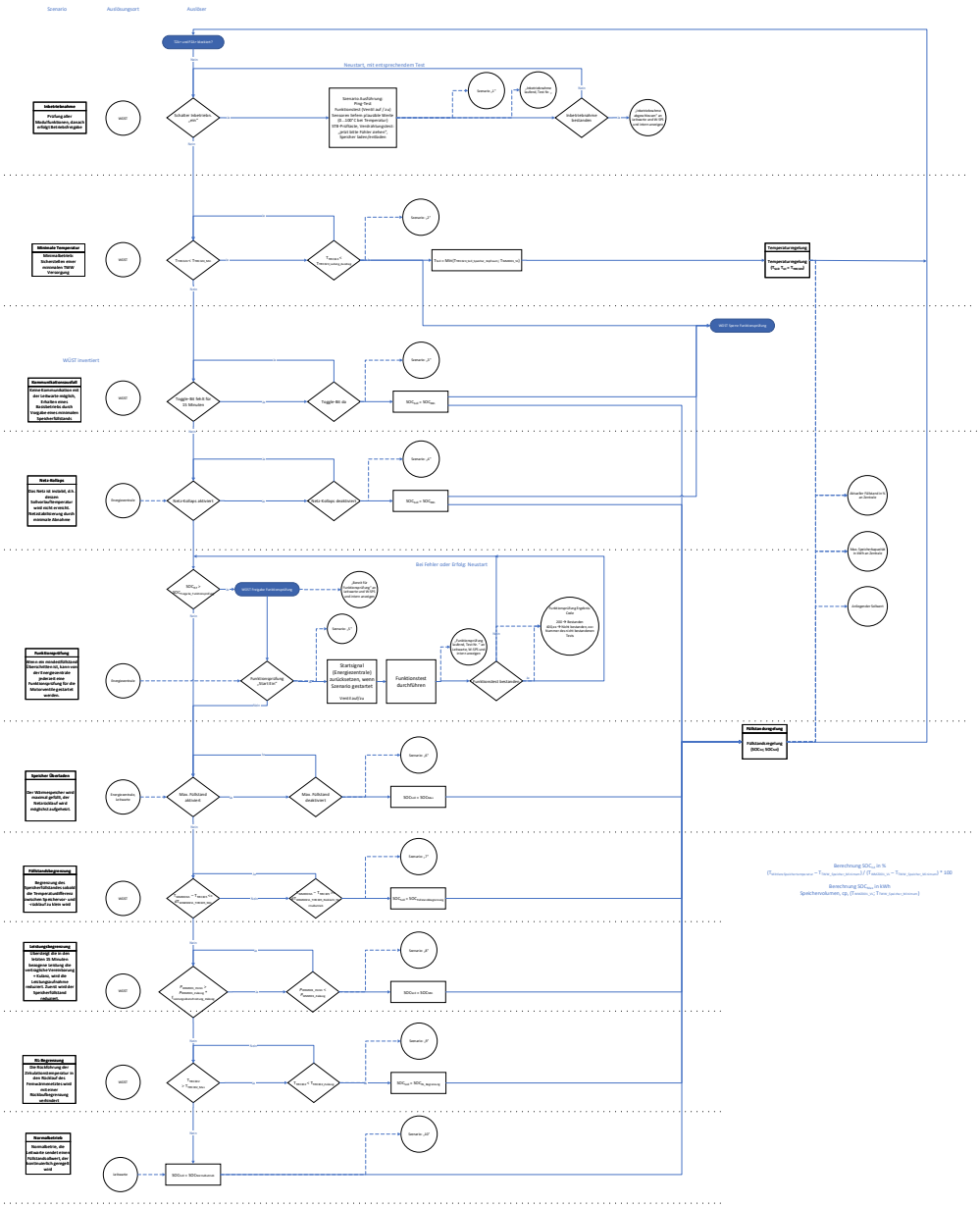


Abbildung A.3.: Regelbeschreibung iWÜST mit TWW

Dictionary und Quellcodes

Quellcode A.1: Dictionary “codesys.info” des Steuerungsprogramms, erstellt durch CODESYSImporter (exklusive der Anwenderprogramme)

```
program_list_pid:
- PID_Pviernulleins
- PID_generator
- PID_MVdreinulleins
- PID_MVeinsnulleins
plc_clamps_dict:
  '''0750-0460/0000-0003 4 AI PT1000 (RTD)''':
    PARAM 10001: '''plugged'''
    channels_clamp:
    - '''Ch_1 signed Input value'''
    - '''Ch_2 signed Input value'''
    - '''Ch_3 signed Input value'''
    - '''Ch_4 signed Input value'''
    EXCLUDEFROMAUTOADR: '0'
    '''Ch_3 signed Input value''':
      SECTION_NAME_CHANNEL: '''INTOnW_I'''
      CHANNEL_MODE_CHANNEL: '''I'''
      COMMENT_CHANNEL: '''Ch_3 signed Input value'''
      SYMBOLIC_NAME_CHANNEL: '''TIRC305_clamp'''
      IECADR_CHANNEL: '%IW2'
      INDEX_IN_PARENT_CHANNEL: '''3'''
    SECTION_NAME: '''Type_44_4_Channels'''
    COMMENT: ''''''
    PARAM 10001_num: '0'
    IECDIAG: '%MB0'
    PARAM 10000_num: '0'
    '''Ch_2 signed Input value''':
      SECTION_NAME_CHANNEL: '''INTOnW_I'''
      CHANNEL_MODE_CHANNEL: '''I'''
      COMMENT_CHANNEL: '''Ch_2 signed Input value'''
      SYMBOLIC_NAME_CHANNEL: '''TC101_clamp'''
      IECADR_CHANNEL: '%IW1'
      INDEX_IN_PARENT_CHANNEL: '''2'''
    '''Ch_1 signed Input value''':
      SECTION_NAME_CHANNEL: '''INTOnW_I'''
      CHANNEL_MODE_CHANNEL: '''I'''
      COMMENT_CHANNEL: '''Ch_1 signed Input value'''
      SYMBOLIC_NAME_CHANNEL: '''TIRC103_clamp'''
      IECADR_CHANNEL: '%IW0'
      INDEX_IN_PARENT_CHANNEL: '''1'''
    DOWNLOAD: '1'
    INDEX_IN_PARENT: '''3'''
    IECOUT: '%QB16'
    NODE_ID: '2'
    '''Ch_4 signed Input value''':
      SECTION_NAME_CHANNEL: '''INTOnW_I'''
      CHANNEL_MODE_CHANNEL: '''I'''
      COMMENT_CHANNEL: '''Ch_4 signed Input value'''
      SYMBOLIC_NAME_CHANNEL: '''TIRC102_clamp'''
      IECADR_CHANNEL: '%IW3'
      INDEX_IN_PARENT_CHANNEL: '''4'''
    PARAM 10000: '''PLC'''
    num_of_channels: 4
    IECIN: '%IB16'
```

```
55      MODULE: '''3S'''
56    '''0750-0460/0000-0003 4 AI PT1000 (RTD)''5':
57      PARAM 10001: '''plugged'''
58      channels_clamp:
59      - '''Ch_1 signed Input value'''
60      - '''Ch_2 signed Input value'''
61      - '''Ch_3 signed Input value'''
62      - '''Ch_4 signed Input value'''
63      EXCLUDEFROMAUTOADR: '0'
64       '''Ch_3 signed Input value''':
65         SECTION_NAME_CHANNEL: '''INTOnW_I'''
66         CHANNEL_MODE_CHANNEL: '''I'''
67         COMMENT_CHANNEL: '''Ch_3 signed Input value'''
68         SYMBOLIC_NAME_CHANNEL: '''TIRC302_clamp'''
69         IECADR_CHANNEL: '%IW6'
70         INDEX_IN_PARENT_CHANNEL: '''3'''
71      SECTION_NAME: '''Type_44_4_Channels'''
72      COMMENT: ''''''
73      PARAM 10001_num: '0'
74      IECDIAG: '%MB0'
75      PARAM 10000_num: '0'
76       '''Ch_2 signed Input value''':
77         SECTION_NAME_CHANNEL: '''INTOnW_I'''
78         CHANNEL_MODE_CHANNEL: '''I'''
79         COMMENT_CHANNEL: '''Ch_2 signed Input value'''
80         SYMBOLIC_NAME_CHANNEL: '''storage2_TIRC320'''
81         IECADR_CHANNEL: '%IW5'
82         INDEX_IN_PARENT_CHANNEL: '''2'''
83       '''Ch_1 signed Input value''':
84         SECTION_NAME_CHANNEL: '''INTOnW_I'''
85         CHANNEL_MODE_CHANNEL: '''I'''
86         COMMENT_CHANNEL: '''Ch_1 signed Input value'''
87         SYMBOLIC_NAME_CHANNEL: '''TIR301_clamp'''
88         IECADR_CHANNEL: '%IW4'
89         INDEX_IN_PARENT_CHANNEL: '''1'''
90      DOWNLOAD: '1'
91      INDEX_IN_PARENT: '''6'''
92      IECOUT: '%QB24'
93      NODE_ID: '5'
94       '''Ch_4 signed Input value''':
95         SECTION_NAME_CHANNEL: '''INTOnW_I'''
96         CHANNEL_MODE_CHANNEL: '''I'''
97         COMMENT_CHANNEL: '''Ch_4 signed Input value'''
98         SYMBOLIC_NAME_CHANNEL: '''TIRC401_clamp'''
99         IECADR_CHANNEL: '%IW7'
100        INDEX_IN_PARENT_CHANNEL: '''4'''
101     PARAM 10000: '''PLC'''
102     num_of_channels: 4
103     IECIN: '%IB24'
104     MODULE: '''3S'''
105   '''0750-0559 4 AO 0-10V DC''':
106     PARAM 10001: '''plugged'''
107     channels_clamp:
108     - '''Ch_1 Analog output'''
109     - '''Ch_2 Analog output'''
110     - '''Ch_3 Analog output'''
111     - '''Ch_4 Analog output'''
112      '''Ch_3 Analog output''':
113        SECTION_NAME_CHANNEL: '''WORDOnW_Q'''
114        CHANNEL_MODE_CHANNEL: '''Q'''
```

```
115         COMMENT_CHANNEL: '''Ch_3 Analog output'''
116         SYMBOLIC_NAME_CHANNEL: ''''''
117         IECADR_CHANNEL: '%QW6'
118         INDEX_IN_PARENT_CHANNEL: '''3'''
119       EXCLUDEFROMAUTOADR: '0'
120       '''Ch_2 Analog output''':
121         SECTION_NAME_CHANNEL: ''WORDOnW_Q'''
122         CHANNEL_MODE_CHANNEL: '''Q'''
123         COMMENT_CHANNEL: '''Ch_2 Analog output'''
124         SYMBOLIC_NAME_CHANNEL: ''''''
125         IECADR_CHANNEL: '%QW5'
126         INDEX_IN_PARENT_CHANNEL: '''2'''
127       SECTION_NAME: '''Type_14_4_Channels'''
128       '''Ch_4 Analog output''':
129         SECTION_NAME_CHANNEL: ''WORDOnW_Q'''
130         CHANNEL_MODE_CHANNEL: '''Q'''
131         COMMENT_CHANNEL: '''Ch_4 Analog output'''
132         SYMBOLIC_NAME_CHANNEL: ''''''
133         IECADR_CHANNEL: '%QW7'
134         INDEX_IN_PARENT_CHANNEL: '''4'''
135       COMMENT: ''''''
136       PARAM 10001_num: '0'
137       IECDIAG: '%MB0'
138       PARAM 10000_num: '0'
139       DOWNLOAD: '1'
140       INDEX_IN_PARENT: '''5'''
141       IECOUT: '%QB20'
142       NODE_ID: '4'
143       PARAM 10000: '''PLC'''
144       num_of_channels: 4
145       IECIN: '%IB24'
146       '''Ch_1 Analog output''':
147         SECTION_NAME_CHANNEL: ''WORDOnW_Q'''
148         CHANNEL_MODE_CHANNEL: '''Q'''
149         COMMENT_CHANNEL: '''Ch_1 Analog output'''
150         SYMBOLIC_NAME_CHANNEL: '''P401_clamp'''
151         IECADR_CHANNEL: '%QW4'
152         INDEX_IN_PARENT_CHANNEL: '''1'''
153       MODULE: '''3S'''
154     '''0750-0554 2 AO 4-20mA''':
155       PARAM 10001: '''plugged'''
156       channels_clamp:
157       - '''Ch_1 Analog output'''
158       - '''Ch_2 Analog output'''
159       EXCLUDEFROMAUTOADR: '0'
160       '''Ch_2 Analog output''':
161         SECTION_NAME_CHANNEL: ''WORDOnW_Q'''
162         CHANNEL_MODE_CHANNEL: '''Q'''
163         COMMENT_CHANNEL: '''Ch_2 Analog output'''
164         SYMBOLIC_NAME_CHANNEL: '''MV301_clamp'''
165         IECADR_CHANNEL: '%QW1'
166         INDEX_IN_PARENT_CHANNEL: '''2'''
167       SECTION_NAME: '''Type_14_2_Channels'''
168       COMMENT: ''''''
169       PARAM 10001_num: '0'
170       IECDIAG: '%MB0'
171       PARAM 10000_num: '0'
172       DOWNLOAD: '1'
173       INDEX_IN_PARENT: '''1'''
174       IECOUT: '%QB16'
```

```
    NODE_ID: '0'
    PARAM 10000: '''PLC'''
    num_of_channels: 2
    IECIN: '%IB16'
    '''Ch_1 Analog output''':
      SECTION_NAME_CHANNEL: '''WORDOnW_Q'''
      CHANNEL_MODE_CHANNEL: '''Q'''
      COMMENT_CHANNEL: '''Ch_1 Analog output'''
      SYMBOLIC_NAME_CHANNEL: '''MV101_clamp'''
      IECADR_CHANNEL: '%QW0'
      INDEX_IN_PARENT_CHANNEL: '''1'''
    MODULE: '''3S'''
  '''0750-0460/0000-0003 4 AI PT1000 (RTD)''6':
    PARAM 10001: '''plugged'''
    channels_clamp:
    - '''Ch_1 signed Input value'''
    - '''Ch_2 signed Input value'''
    - '''Ch_3 signed Input value'''
    - '''Ch_4 signed Input value'''
    EXCLUDEFROMAUTOADR: '0'
    '''Ch_3 signed Input value''':
      SECTION_NAME_CHANNEL: '''INTOnW_I'''
      CHANNEL_MODE_CHANNEL: '''I'''
      COMMENT_CHANNEL: '''Ch_3 signed Input value'''
      SYMBOLIC_NAME_CHANNEL: '''storage2_TIRC321'''
      IECADR_CHANNEL: '%IW10'
      INDEX_IN_PARENT_CHANNEL: '''3'''
    SECTION_NAME: '''Type_44_4_Channels'''
    COMMENT: ''''''
    PARAM 10001_num: '0'
    IECDIAG: '%MB0'
    PARAM 10000_num: '0'
    '''Ch_2 signed Input value''':
      SECTION_NAME_CHANNEL: '''INTOnW_I'''
      CHANNEL_MODE_CHANNEL: '''I'''
      COMMENT_CHANNEL: '''Ch_2 signed Input value'''
      SYMBOLIC_NAME_CHANNEL: '''storage1_TIRC324'''
      IECADR_CHANNEL: '%IW9'
      INDEX_IN_PARENT_CHANNEL: '''2'''
    '''Ch_1 signed Input value''':
      SECTION_NAME_CHANNEL: '''INTOnW_I'''
      CHANNEL_MODE_CHANNEL: '''I'''
      COMMENT_CHANNEL: '''Ch_1 signed Input value'''
      SYMBOLIC_NAME_CHANNEL: '''storage1_TIRC322'''
      IECADR_CHANNEL: '%IW8'
      INDEX_IN_PARENT_CHANNEL: '''1'''
    DOWNLOAD: '1'
    INDEX_IN_PARENT: '''7'''
    IECOUT: '%QB32'
    NODE_ID: '6'
    '''Ch_4 signed Input value''':
      SECTION_NAME_CHANNEL: '''INTOnW_I'''
      CHANNEL_MODE_CHANNEL: '''I'''
      COMMENT_CHANNEL: '''Ch_4 signed Input value'''
      SYMBOLIC_NAME_CHANNEL: '''storage1_TIRC323'''
      IECADR_CHANNEL: '%IW11'
      INDEX_IN_PARENT_CHANNEL: '''4'''
    PARAM 10000: '''PLC'''
    num_of_channels: 4
    IECIN: '%IB24'
```

```
235       MODULE: '''3S'''
236     '''0750-0400 2 DI 24 V DC 3.0ms''':
237       PARAM 10001: '''plugged'''
238       channels_clamp:
239       - '''Ch_1 Digital input'''
240       - '''Ch_2 Digital input'''
241       EXCLUDEFROMAUTOADR: '0'
242       SECTION_NAME: '''Type_1_2_Channels'''
243       '''Ch_2 Digital input''':
244         SECTION_NAME_CHANNEL: '''BOOLOnX_I'''
245         CHANNEL_MODE_CHANNEL: '''I'''
246         COMMENT_CHANNEL: '''Ch_2 Digital input'''
247         SYMBOLIC_NAME_CHANNEL: '''TZ_clamp'''
248         IECADR_CHANNEL: '%IX12.1'
249         INDEX_IN_PARENT_CHANNEL: '''2'''
250       COMMENT: ''''''
251       PARAM 10001_num: '0'
252       IECDIAG: '%MB0'
253       PARAM 10000_num: '0'
254       DOWNLOAD: '1'
255       INDEX_IN_PARENT: '''2'''
256       IECOUT: '%QB16'
257       NODE_ID: '1'
258       PARAM 10000: '''PLC'''
259       num_of_channels: 2
260       IECIN: '%IB24'
261       '''Ch_1 Digital input''':
262         SECTION_NAME_CHANNEL: '''BOOLOnX_I'''
263         CHANNEL_MODE_CHANNEL: '''I'''
264         COMMENT_CHANNEL: '''Ch_1 Digital input'''
265         SYMBOLIC_NAME_CHANNEL: '''TS_clamp'''
266         IECADR_CHANNEL: '%IX12.0'
267         INDEX_IN_PARENT_CHANNEL: '''1'''
268       MODULE: '''3S'''
269     '''0750-0554 2 AO 4-20mA''3':
270 ␣␣␣␣PARAM␣10001:␣'''plugged'''
271       channels_clamp:
272       - '''Ch_1 Analog output'''
273       - '''Ch_2 Analog output'''
274       EXCLUDEFROMAUTOADR: '0'
275       '''Ch_2 Analog output''':
276         SECTION_NAME_CHANNEL: '''WORDOnW_Q'''
277         CHANNEL_MODE_CHANNEL: '''Q'''
278         COMMENT_CHANNEL: '''Ch_2 Analog output'''
279         SYMBOLIC_NAME_CHANNEL: ''''''
280         IECADR_CHANNEL: '%QW3'
281         INDEX_IN_PARENT_CHANNEL: '''2'''
282       SECTION_NAME: '''Type_14_2_Channels'''
283       COMMENT: ''''''
284       PARAM 10001_num: '0'
285       IECDIAG: '%MB0'
286       PARAM 10000_num: '0'
287       DOWNLOAD: '1'
288       INDEX_IN_PARENT: '''4'''
289       IECOUT: '%QB20'
290       NODE_ID: '3'
291       PARAM 10000: '''PLC'''
292       num_of_channels: 2
293       IECIN: '%IB16'
294       '''Ch_1 Analog output''':
```

```
295         SECTION_NAME_CHANNEL: '''WORDOnW_Q'''
296         CHANNEL_MODE_CHANNEL: '''Q'''
297         COMMENT_CHANNEL: '''Ch_1 Analog output'''
298         SYMBOLIC_NAME_CHANNEL: '''MV302_clamp'''
299         IECADR_CHANNEL: '%QW2'
300         INDEX_IN_PARENT_CHANNEL: '''1'''
301       MODULE: '''3S'''
302   main_loop_programs:
303   - PLC_PRG_TWW
304   - SETOUTPUT
305   - PLC_PRG_WUEST
306   programs_dict:
307     PID_Pviernulleins: "VAR\n          pid_P401: PID;\nEND_VAR\n(* @END_DECLARATION :=\
308       \ '0' *)\n_FBD_BODY\n_NETWORKS : 1\n_NETWORK\n\n_COMMENT\n''\n_END_COMMENT\n_ASSIGN\n\
309       _FUNCTIONBLOCK\npid_P401\n_BOX_EXPR : 11\n_OPERAND\n_EXPRESSION\n_POSITIV\nTIRC401_clamp_\n\
310       _OPERAND\n_EXPRESSION\n_POSITIV\nTIRC401_clamp_soll\n_OPERAND\n_EXPRESSION\n_POSITIV\n\
311       0.00005\n_OPERAND\n_EXPRESSION\n_POSITIV\n10000000\n_OPERAND\n_EXPRESSION\n_POSITIV\n\
312       0\n_OPERAND\n_EXPRESSION\n_POSITIV\n0\n_OPERAND\n_EXPRESSION\n_POSITIV\n0\n_OPERAND\n\
313       _EXPRESSION\n_POSITIV\n-0.02\n_OPERAND\n_EXPRESSION\n_POSITIV\n0.02\n_OPERAND\n\
314       _EXPRESSION\n_POSITIV\nFALSE\n_OPERAND\n_EXPRESSION\n_POSITIV\nFALSE\n_EXPRESSION\n\
315       _POSITIV\nPID\n_OUTPUTS : 2\n_OUTPUT\n_POSITIV\n_NO_SET\n_EMPTY\n_OUTPUT\n_POSITIV\n\
316       _NO_SET\n_EMPTY\n_EXPRESSION\n_POSITIV\n_OUTPUTS : 1\n_OUTPUT\n_POSITIV\n_NO_SET\n\
317       P401_clamp_\n"
318     PLC_PRG_TWW: ---Code des Anwenderprogramms PLC_PRG_TWW()---
319     PID_generator: "VAR\n          pid_generator: PID;\nEND_VAR\n(* @END_DECLARATION :=\
320       \ '0' *)\n_FBD_BODY\n_NETWORKS : 1\n_NETWORK\n\n_COMMENT\n''\n_END_COMMENT\n_ASSIGN\n\
321       _FUNCTIONBLOCK\npid_generator\n_BOX_EXPR : 11\n_OPERAND\n_EXPRESSION\n_POSITIV\n\
322       T_Netz_ist\n_OPERAND\n_EXPRESSION\n_POSITIV\nT_Netz_soll\n_OPERAND\n_EXPRESSION\n\
323       _POSITIV\n0.00005\n_OPERAND\n_EXPRESSION\n_POSITIV\n10000000\n_OPERAND\n_EXPRESSION\n\
324       _POSITIV\n0\n_OPERAND\n_EXPRESSION\n_POSITIV\n0\n_OPERAND\n_EXPRESSION\n_POSITIV\n\
325       0\n_OPERAND\n_EXPRESSION\n_POSITIV\n-0.02\n_OPERAND\n_EXPRESSION\n_POSITIV\n0.02\n\
326       _OPERAND\n_EXPRESSION\n_POSITIV\nFALSE\n_OPERAND\n_EXPRESSION\n_POSITIV\nFALSE\n\
327       _EXPRESSION\n_POSITIV\nPID\n_OUTPUTS : 2\n_OUTPUT\n_POSITIV\n_NO_SET\n_EMPTY\n\
328       _OUTPUT\n_POSITIV\n_NO_SET\n_EMPTY\n_EXPRESSION\n_POSITIV\n_OUTPUTS : 1\n_OUTPUT\n\
329       _POSITIV\n_NO_SET\ngenerator_clamp_\n"
330     PID_MVdreinulleins: "VAR\n          pid_MV301: PID;\nEND_VAR\n(* @END_DECLARATION\
331       \ := '0' *)\n_FBD_BODY\n_NETWORKS : 1\n_NETWORK\n\n_COMMENT\n''\n_END_COMMENT\n\
332       _ASSIGN\n_FUNCTIONBLOCK\npid_MV301\n_BOX_EXPR : 11\n_OPERAND\n_EXPRESSION\n_POSITIV\n\
333       MV301_ist\n_OPERAND\n_EXPRESSION\n_POSITIV\nMV301_soll\n_OPERAND\n_EXPRESSION\n\
334       _POSITIV\n0.000025\n_OPERAND\n_EXPRESSION\n_POSITIV\n10000000\n_OPERAND\n_EXPRESSION\n\
335       _POSITIV\n0\n_OPERAND\n_EXPRESSION\n_POSITIV\n0\n_OPERAND\n_EXPRESSION\n_POSITIV\n\
336       0\n_OPERAND\n_EXPRESSION\n_POSITIV\n-0.02\n_OPERAND\n_EXPRESSION\n_POSITIV\n0.02\n\
337       _OPERAND\n_EXPRESSION\n_POSITIV\nFALSE\n_OPERAND\n_EXPRESSION\n_POSITIV\nFALSE\n\
338       _EXPRESSION\n_POSITIV\nPID\n_OUTPUTS : 2\n_OUTPUT\n_POSITIV\n_NO_SET\n_EMPTY\n\
339       _OUTPUT\n_POSITIV\n_NO_SET\n_EMPTY\n_EXPRESSION\n_POSITIV\n_OUTPUTS : 1\n_OUTPUT\n\
340       _POSITIV\n_NO_SET\nMV301_clamp_\n"
341     PID_MVeinsnulleins: "VAR\n          pid_MV101: PID;\nEND_VAR\n(* @END_DECLARATION\
342       \ := '0' *)\n_FBD_BODY\n_NETWORKS : 1\n_NETWORK\n\n_COMMENT\n''\n_END_COMMENT\n\
343       _ASSIGN\n_FUNCTIONBLOCK\npid_MV101\n_BOX_EXPR : 11\n_OPERAND\n_EXPRESSION\n_POSITIV\n\
344       MV101_ist\n_OPERAND\n_EXPRESSION\n_POSITIV\nMV101_soll\n_OPERAND\n_EXPRESSION\n\
345       _POSITIV\nKp_MV101\n_OPERAND\n_EXPRESSION\n_POSITIV\nTn_MV101\n_OPERAND\n_EXPRESSION\n\
346       _POSITIV\nTv_MV101\n_OPERAND\n_EXPRESSION\n_POSITIV\n0\n_OPERAND\n_EXPRESSION\n\
347       _POSITIV\n0\n_OPERAND\n_EXPRESSION\n_POSITIV\n-0.02\n_OPERAND\n_EXPRESSION\n_POSITIV\n\
348       0.02\n_OPERAND\n_EXPRESSION\n_POSITIV\nFALSE\n_OPERAND\n_EXPRESSION\n_POSITIV\n\
349       FALSE\n_EXPRESSION\n_POSITIV\nPID\n_OUTPUTS : 2\n_OUTPUT\n_POSITIV\n_NO_SET\n\
350       _EMPTY\n_OUTPUT\n_POSITIV\n_NO_SET\n_EMPTY\n_EXPRESSION\n_POSITIV\n_OUTPUTS :\
351       \ 1\n_OUTPUT\n_POSITIV\n_NO_SET\nMV101_clamp_\n"
352     SETOUTPUT: ---Code des Anwenderprogramms SETOUTPUT()---
353     MODBUS_CONFIGURATION: "VAR                                    \nEND_VAR          \
354       \                \n(* @END_DECLARATION := '0' *)\n(*                          \
```

```
355     \          \n    <?xml version=\"1.0\" encoding=\"UTF-16\" standalone=\"yes\"?>\n\
356     <network xml_structure_version=\"1\" xml_feature_version=\"1\">\n          <generator_settings\
357     \ minRTUTaskCycleTimeMs=\"5\" minEthaskCycleTimeMs=\"5\" ethSlaveMultiplier=\"\
358     0.1\" rtuInterfaceMultiplier=\"0.1\" taskGeneration=\"false\"/>\n</network>\n\
359     \                                  \n*)                                        \n;       \
360     \                                "
361   PLC_PRG_WUEST: ---Code des Anwenderprogramms PLC_PRG_WUEST()---
362 pid_dict:
363   pid_generator:
364     Y_OFFSET: '0'
365     KP: '0.00005'
366     TN: '10000000'
367     Y: generator_clamp_
368     MANUAL: 'FALSE'
369     RESET: 'FALSE'
370     TV: '0'
371     Y_MAX: '0.02'
372     ACTUAL: T_Netz_ist
373     SET_POINT: T_Netz_soll
374     LIMITS_ACTIVE: _EMPTY
375     OVERFLOW: _EMPTY
376     Y_MANUAL: '0'
377     Y_MIN: '-0.02'
378   pid_P401:
379     Y_OFFSET: '0'
380     KP: '0.00005'
381     TN: '10000000'
382     Y: P401_clamp_
383     MANUAL: 'FALSE'
384     RESET: 'FALSE'
385     TV: '0'
386     Y_MAX: '0.02'
387     ACTUAL: TIRC401_clamp_
388     SET_POINT: TIRC401_clamp_soll
389     LIMITS_ACTIVE: _EMPTY
390     OVERFLOW: _EMPTY
391     Y_MANUAL: '0'
392     Y_MIN: '-0.02'
393   pid_MV101:
394     Y_OFFSET: '0'
395     KP: Kp_MV101
396     TN: Tn_MV101
397     Y: MV101_clamp_
398     MANUAL: 'FALSE'
399     RESET: 'FALSE'
400     TV: Tv_MV101
401     Y_MAX: '0.02'
402     ACTUAL: MV101_ist
403     SET_POINT: MV101_soll
404     LIMITS_ACTIVE: _EMPTY
405     OVERFLOW: _EMPTY
406     Y_MANUAL: '0'
407     Y_MIN: '-0.02'
408   pid_MV301:
409     Y_OFFSET: '0'
410     KP: '0.000025'
411     TN: '10000000'
412     Y: MV301_clamp_
413     MANUAL: 'FALSE'
414     RESET: 'FALSE'
```

```
415       TV: '0'
416       Y_MAX: '0.02'
417       ACTUAL: MV301_ist
418       SET_POINT: MV301_soll
419       LIMITS_ACTIVE: _EMPTY
420       OVERFLOW: _EMPTY
421       Y_MANUAL: '0'
422       Y_MIN: '-0.02'
423   plc_info_dict:
424     '''Hardware configuration''':
425       DOWNLOAD: '1'
426       EXCLUDEFROMAUTOADR: '0'
427       INDEX_IN_PARENT: '''-1'''
428       IECOUT: '%QB0'
429       COMMENT: ''''''
430       SECTION_NAME: '''Root'''
431       MODULE: '''3S'''
432       IECIN: '%IB0'
433       NODE_ID: '-1'
434       IECDIAG: '%MB0'
435     '''Flag variables''':
436       DOWNLOAD: '1'
437       EXCLUDEFROMAUTOADR: '0'
438       INDEX_IN_PARENT: '''3'''
439       IECOUT: '%QB0'
440       COMMENT: ''''''
441       SECTION_NAME: '''FLAG_VARS'''
442       MODULE: '''3S'''
443       IECIN: '%IB0'
444       NODE_ID: '2'
445       IECDIAG: '%MB0'
446     '''Fieldbus variables''':
447       DOWNLOAD: '1'
448       EXCLUDEFROMAUTOADR: '0'
449       INDEX_IN_PARENT: '''2'''
450       IECOUT: '%QB0'
451       COMMENT: ''''''
452       SECTION_NAME: '''FB_VARS'''
453       MODULE: '''3S'''
454       IECIN: '%IB0'
455       NODE_ID: '1'
456       IECDIAG: '%MB0'
457     '''Modbus-Master''':
458   ␣␣␣␣DOWNLOAD:␣'1'
459   ␣␣␣␣EXCLUDEFROMAUTOADR:␣'0'
460   ␣␣␣␣INDEX_IN_PARENT:␣'''4'''
461       IECOUT: '%QB0'
462       COMMENT: ''''''
463       SECTION_NAME: '''MB_MASTER'''
464       MODULE: '''3S'''
465       IECIN: '%IB0'
466       NODE_ID: '3'
467       IECDIAG: '%MB0'
468     '''K-Bus''':
469       DOWNLOAD: '1'
470       EXCLUDEFROMAUTOADR: '0'
471       INDEX_IN_PARENT: '''1'''
472       IECOUT: '%QB0'
473       COMMENT: '''Ethernet Controller 100MBit 2Port'''
474       SECTION_NAME: '''K_Bus'''
```

```
475       MODULE:  ' ' '3S ' ' '
476       IECIN: '%IB0'
477       NODE_ID: '0'
478       IECDIAG: '%MB0'
479   pid_prg_dict:
480     PID_Pviernulleins:
481     - pid_P401
482     PID_MVdreinulleins:
483     - pid_MV301
484     PID_MVeinsnulleins:
485     - pid_MV101
486     PID_generator:
487     - pid_generator
488   tasks_dict:
489     TaskPID:
490       PRIORITY: '3'
491       PROGRAM_CALL3: PID_MVeinsnulleins
492       PROGRAM_CALL1: PID_generator
493       NUM_OF_CALLS: 4
494       TYP: cyclical
495       PROGRAM_CALL2: PID_MVdreinulleins
496       PROGRAM_CALL4: PID_Pviernulleins
497       INTERVAL: 200ms
498     TaskSETOUTPUT:
499       PRIORITY: '4'
500       NUM_OF_CALLS: 1
501       INTERVAL: 200ms
502       TYP: cyclical
503       PROGRAM_CALL1: SETOUTPUT
504     TaskTWW:
505       PRIORITY: '1'
506       NUM_OF_CALLS: 1
507       INTERVAL: 200ms
508       TYP: cyclical
509       PROGRAM_CALL1: PLC_PRG_TWW
510     TaskWUEST:
511       PRIORITY: '2'
512       NUM_OF_CALLS: 1
513       INTERVAL: 200ms
514       TYP: cyclical
515       PROGRAM_CALL1: PLC_PRG_WUEST
516   plc_global_vars:
517     generator_clamp_: 0
518     T_Netz_ist: 0
519     TZ_clamp: 'False'
520     STATUS_funktionspruefung_WUE: '0'
521     TIRC_min: '0'
522     FLAG_2_TWW: 0
523     Zaehler_WUE_9: 0
524     MV301_clamp_: 0
525     FLAG_Timer_Fuellstandsbegrenzung: 0
526     c_p: '4190'
527     T_Netz_soll: 0
528     P401_clamp_: 0
529     MV301_soll: 0
530     P_WMZ001_zulaessig: '105'
531     FLAG_2_WUE: 0
532     TIRC401_clamp_soll: '60'
533     STATUS_funktionspruefung_TWW: '0'
534     FLAG_3: 0
```

```
SOC: 0
Szenario_WUE_alt: 0
EXTERN_Toggle: 'True'
TIRC401_clamp_: 0
MV301_ist: 0
EXTERN_Netzkollaps: 'False'
Skalierung_Temp_clamp: '1'
MV101_ist: 0
TIRC_max: '100'
MV101_soll: 0
temp_sensor_help_clamp_: 0
Szenario_WUE: 0
TC101_clamp_: 0
Zaehler_gesamt: 0
TIRC301_clamp_: 0
generator_clamp: 0
P_Heizkreise: 0
Szenario_TWW: 0
FLAG_9_WUE: 0
T_Netz_mittel: '70'
TIRC305_clamp_: 0
MV302_clamp_: 0
OFFSET_Kelvin: '273.15'
Ergebnis_funktionspruefung_TWW: 0
FLAG_JARVIS: 'True'
select_signal: 0
FLAG_0_WUE: 0
MV101_clamp_: 0
EXTERN_DO_funktionspruefung_WUE: 'False'
P_WMZ001: 0
start_funktionspruefung_1_TWW: 0
TS_clamp: 'False'
```

Quellcode A.2: Token/Sequenzen des STParsers

```
INDENT = "␣␣␣␣"
NEWLINE = "\n"
TK_WHITE = pp.White(min=1)
TK_REAL = pp.Optional("(") + pp.Combine(pp.Optional("-") + pp.Word(pp.nums) + pp.Optional("." +
          pp.Word(pp.nums))) + pp.Optional(")")
TK_COLON = (pp.Optional(TK_WHITE) + pp.Literal(":") +
            pp.Optional(TK_WHITE)).setParseAction(pp.replaceWith(""))
TK_SEMICOLON = pp.Literal(";").setParseAction(pp.replaceWith(""))
TK_SEMICOLON_REPEAT = pp.Literal(";").setParseAction(pp.replaceWith("\n"))
TK_VARIABLE = pp.Optional(pp.Word("(")) + pp.Combine(pp.Optional("-") + pp.Word(pp.alphanums +
              "_" + "." + "[" + "]" + "'")) + pp.Optional(pp.Word(")"))
TK_MODULO = pp.CaselessLiteral("MOD").setParseAction(pp.replaceWith("%"))
TK_OP_ARITH = (pp.Word("-" + "+" + "*" + "/" + "␣-␣" + "␣+␣" + "␣*␣" + "␣/␣") | TK_MODULO)
TK_TIMER_CALL = pp.Word(pp.alphanums + "_") + "." + (pp.Literal("TON(") | pp.Literal("TOF(") |
                pp.Literal("TP(")) + pp.Word(pp.alphanums + "_") + ")" + "[" + pp.Word("01") + "]"
TK_TIME = (pp.CaselessLiteral("T#") | pp.CaselessLiteral("TIME#")) + pp.OneOrMore(pp.Word(pp.nums)
          + pp.Word(pp.alphas, max=2)) + (TK_SEMICOLON | pp.Literal(")") | pp.White(min=1))
SQ_REFERENCE_VALUE = ((TK_TIMER_CALL | TK_REAL | TK_VARIABLE | TK_TIME) +
                      pp.Optional(pp.OneOrMore(TK_OP_ARITH + (TK_REAL | TK_VARIABLE))))
                      ^ ((TK_REAL | TK_VARIABLE) + pp.Word("(") + (TK_REAL | TK_VARIABLE) +
                      TK_OP_ARITH + (TK_REAL | TK_VARIABLE) + pp.Word(")"))
TK_EQUAL = (pp.Literal(":=") + pp.Optional(TK_WHITE)).setParseAction(pp.replaceWith("=␣"))
TK_AND = pp.White(min=1) + (pp.CaselessLiteral("and") |
         pp.Word("&")).setParseAction(pp.replaceWith("and"))
TK_OR = pp.White(min=1) + pp.CaselessLiteral("or").setParseAction(pp.replaceWith("or"))
```

```
26      TK_XOR = pp.White(min=1) + pp.CaselessLiteral("xor").setParseAction(pp.replaceWith("xor"))
27      TK_COMMA = (pp.Optional(TK_WHITE) + pp.Literal(",") +
28                  pp.Optional(TK_WHITE)).setParseAction(pp.replaceWith(","))
29      TK_LOGICAL = TK_AND | TK_OR | TK_XOR
30      TK_INT = pp.CaselessLiteral("␣int(").setParseAction(pp.replaceWith("round("))
31      TK_ROUND = pp.CaselessLiteral("round(") + pp.Word(pp.nums) + pp.Literal(".") +
32                 pp.Literal("5)").setParseAction(pp.replaceWith("50000001)"))
33      TK_TRUNC = pp.CaselessLiteral("trunc(").setParseAction(pp.replaceWith("int("))
34      TK_FALSE = pp.CaselessLiteral("False")
35      TK_TRUE = pp.CaselessLiteral("True")
36      TK_VECTOR_ELEMENT = pp.Literal("[") + pp.Word(pp.nums) + pp.Literal("]")
37      TK_NOT = pp.Word("+−*/()␣") + pp.CaselessLiteral("NOT␣").setParseAction(pp.replaceWith("~"))
38      TK_ABS = (TK_OP_ARITH | pp.Literal("=")) + pp.CaselessLiteral("abs")
39      TK_SQRT = (TK_OP_ARITH | pp.Literal("=")) +
40                pp.CaselessLiteral("sqrt(").setParseAction(pp.replaceWith("math.sqrt("))
41      TK_LN = (TK_OP_ARITH | pp.Literal("=")) +
42              pp.CaselessLiteral("ln(").setParseAction(pp.replaceWith("math.log("))
43      TK_LOG = (TK_OP_ARITH | pp.Literal("=")) +
44               pp.CaselessLiteral("log(").setParseAction(pp.replaceWith("math.log10("))
45      TK_EXP = (TK_OP_ARITH | pp.Literal("=")) +
46               pp.CaselessLiteral("exp(").setParseAction(pp.replaceWith("math.exp("))
47      TK_SIN = (TK_OP_ARITH | pp.Literal("=")) +
48               pp.CaselessLiteral("sin(").setParseAction(pp.replaceWith("math.sin("))
49      TK_COS = (TK_OP_ARITH | pp.Literal("=")) +
50               pp.CaselessLiteral("cos(").setParseAction(pp.replaceWith("math.cos("))
51      TK_TAN = (TK_OP_ARITH | pp.Literal("=")) +
52               pp.CaselessLiteral("tan(").setParseAction(pp.replaceWith("math.tan("))
53      TK_ASIN = (TK_OP_ARITH | pp.Literal("=")) +
54               pp.CaselessLiteral("asin(").setParseAction(pp.replaceWith("math.asin("))
55      TK_ACOS = (TK_OP_ARITH | pp.Literal("=")) +
56               pp.CaselessLiteral("acos(").setParseAction(pp.replaceWith("math.acos("))
57      TK_ATAN = (TK_OP_ARITH | pp.Literal("=")) +
58               pp.CaselessLiteral("atan(").setParseAction(pp.replaceWith("math.atan("))
59      TK_SHR = TK_VARIABLE + pp.Literal(":=") + pp.CaselessLiteral("shr(") + pp.Word(pp.alphanums) +
60               TK_COMMA + pp.Word(pp.alphanums) + ")"
61      TK_SHL = TK_VARIABLE + pp.Literal(":=") +  pp.CaselessLiteral("shl(") + pp.Word(pp.alphanums) +
62               TK_COMMA + pp.Word(pp.alphanums) + ")"
63      TK_ROR = TK_VARIABLE + pp.Literal(":=") +  pp.CaselessLiteral("ror(") + pp.Word(pp.alphanums) +
64               TK_COMMA + pp.Word(pp.alphanums) + ")"
65      TK_ROL = TK_VARIABLE + pp.Literal(":=") + pp.CaselessLiteral("rol(") + pp.Word(pp.alphanums) +
66               TK_COMMA + pp.Word(pp.alphanums) + ")"
67      TK_ADR = (TK_OP_ARITH | pp.Literal("=")) +
68               pp.CaselessLiteral("ADR(").setParseAction(pp.replaceWith("id("))
69      TK_MAX = (TK_OP_ARITH | (pp.Literal("="))) + pp.CaselessLiteral("max(")
70      TK_MIN = (TK_OP_ARITH | (pp.Literal("="))) + pp.CaselessLiteral("min(")
71      TK_END_DECLARATION = pp.Literal("(*␣@END_DECLARATION␣=␣'0'␣*)").\
72              setParseAction((pp.replaceWith("")))
73      TK_IF = pp.ZeroOrMore(pp.White(exact=4)) +
74              pp.CaselessLiteral("if␣").setParseAction(pp.replaceWith("\nif"))
75      TK_IF.setDefaultWhitespaceChars("")
76      TK_ELSE_IF = pp.ZeroOrMore(pp.White(exact=4)) +
77              pp.CaselessLiteral("elsif␣").setParseAction(pp.replaceWith("\nelif"))
78      TK_ELSE_IF.setDefaultWhitespaceChars("")
79      TK_ELIF = pp.ZeroOrMore(pp.White(exact=4)) + pp.CaselessLiteral("elif␣")
80      TK_ELIF.setDefaultWhitespaceChars("")
81      TK_ELSE = pp.CaselessLiteral("else␣").setParseAction(pp.replaceWith("else:␣"))
82      TK_THEN = pp.CaselessLiteral("then").setParseAction(pp.replaceWith(":"))
83      TK_END_IF = pp.CaselessLiteral("end_if").suppress()
84      TK_EXIT = pp.CaselessLiteral("exit").setParseAction(pp.replaceWith("break"))
85      TK_CONTINUE = pp.CaselessLiteral("continue").setParseAction(pp.replaceWith("continue"))
```

```
86  TK_FOR = (pp.lineStart() + pp.CaselessLiteral("for␣")).setParseAction((pp.replaceWith("\nfor")))
87  TK_DO = pp.CaselessLiteral("do").setParseAction(pp.replaceWith("):"))
88  TK_OP_RELATIONAL = (pp.Word("=").setParseAction(pp.replaceWith("==")) | (pp.Word("<") +
89                      pp.Word(">")).setParseAction(
90      pp.replaceWith("!=")) | pp.Word("<" + ">" + ">=" + "<="))
91  SQ_EMPTY_LINE = pp.lineStart() + pp.ZeroOrMore(pp.Word("␣")) + pp.lineEnd()
92  SQ_SET_VALUE_HELPER = (TK_VARIABLE + TK_EQUAL + SQ_REFERENCE_VALUE)
93  TK_CASELIST = pp.Literal("+CASELIST+") + pp.Word(pp.nums)
94  SQ_SET_VALUE_HELPER_CASE = (TK_VARIABLE + pp.Word(":=") + pp.OneOrMore(SQ_REFERENCE_VALUE) +
95                              pp.Literal(";")) | TK_CASELIST
96  SQ_SET_VALUE = pp.ZeroOrMore(pp.White(exact=4)) + (SQ_SET_VALUE_HELPER | TK_EXIT) + TK_SEMICOLON
97  SQ_FOR_ATTRIBUTES = TK_FOR + SQ_SET_VALUE_HELPER + pp.CaselessLiteral("to") + (
98      TK_VARIABLE ^ TK_REAL) + pp.CaselessLiteral("by") + (TK_VARIABLE ^ TK_REAL) + TK_DO
99  TK_END_FOR = pp.CaselessLiteral("end_for").setParseAction(pp.replaceWith(NEWLINE))
100 TK_END_FOR_PARSE = pp.CaselessLiteral("end_for")
101 SQ_COMMENT = pp.nestedExpr("(*", "*)")
102 SQ_VAR_DEFINITION = (pp.nestedExpr("VAR", "END_VAR", )).setParseAction(pp.replaceWith("␣"))
103 SQ_RELATION = SQ_REFERENCE_VALUE + pp.Optional(TK_OP_RELATIONAL + SQ_REFERENCE_VALUE +pp.Optional(
104                pp.OneOrMore(TK_LOGICAL + SQ_REFERENCE_VALUE + TK_OP_RELATIONAL +
105                SQ_REFERENCE_VALUE)))
106 SQ_IF_ELSEIF = ((TK_IF | TK_ELSE_IF) + SQ_RELATION +
107                  (pp.Optional(pp.OneOrMore((pp.Word("and␣AND␣&").setParseAction
108                  (pp.replaceWith("and")) | pp.CaselessLiteral("or") | pp.CaselessLiteral("xor")) +
109                  SQ_RELATION))) + TK_THEN) | ((TK_IF | TK_ELSE_IF) + TK_VARIABLE + pp.Literal("in")
110                  + "␣[" + pp.Word(pp.nums + ",␣" + "-") + "]␣" + TK_THEN)
111 SQ_IF_ELSEIF.setDefaultWhitespaceChars("")
112 SQ_REQUEST = (TK_IF | TK_ELSE_IF) + SQ_RELATION + TK_THEN + pp.OneOrMore(SQ_SET_VALUE) +
113              pp.ZeroOrMore(TK_EXIT + TK_SEMICOLON) + pp.ZeroOrMore(TK_ELSE +
114              (SQ_SET_VALUE_HELPER | TK_EXIT) + TK_SEMICOLON)
115 SQ_IF_CLAUSE = pp.OneOrMore(SQ_REQUEST) + TK_END_IF + TK_SEMICOLON
116 SQ_IF_CLAUSE.ignore(pp.cStyleComment)
117 TK_WHILE = ((pp.lineStart() | pp.OneOrMore(pp.Word("␣"))) +
118             (pp.CaselessLiteral("while␣")).setParseAction(
119     pp.replaceWith("\nwhile(")) | pp.CaselessLiteral("while("))
120 TK_END_WHILE = pp.CaselessLiteral("end_while").setParseAction(pp.replaceWith(""))
121 SQ_WHILE = TK_WHILE + pp.Optional(pp.Word("(")) + SQ_RELATION + pp.Optional(pp.Word(")")) + TK_DO
122 TK_CASE = (pp.lineStart() | pp.OneOrMore(pp.Word("␣"))) + pp.CaselessLiteral("case␣")
123 TK_END_CASE = (pp.lineStart() | pp.OneOrMore(pp.Word("␣"))) +
124               pp.CaselessLiteral("end_case").setParseAction(pp.replaceWith("end_if"))
125 TK_OF = pp.CaselessKeyword("of")
126 SQ_CASE_ATTRIBUTES = TK_CASE + TK_VARIABLE + TK_OF
127 TK_SIGNED_INT = pp.lineStart() + pp.Optional("-") + pp.Word(pp.nums)
128 SQ_CASES = (TK_SIGNED_INT + pp.Word(":") + SQ_SET_VALUE_HELPER_CASE) ^ TK_CASELIST
129 TK_ELSE_CASE = pp.CaselessLiteral("\nelse␣").setParseAction(pp.replaceWith("\nelse:␣"))
130 SQ_REQUEST_VARS = (pp.lineStart() ^ pp.OneOrMore(pp.Word("␣"))) + pp.Combine(pp.Optional("-") +
131                   pp.Word(pp.nums)) + pp.OneOrMore(TK_COMMA + pp.Combine(pp.Optional("-") +
132                   pp.Word(pp.nums))) + pp.restOfLine
133 SQ_REQUEST_NUMERATION = pp.Combine(pp.Optional("-") + pp.Word(pp.nums)) + pp.Optional(TK_COMMA +
134                         pp.Combine(pp.Optional("-") + pp.Word(pp.nums)))+ pp.Word("..", exact=2)
135                         + pp.Combine(pp.Optional("-") + pp.Word(pp.nums)) + pp.restOfLine
136 TK_REPEAT = (pp.lineStart() | pp.OneOrMore(pp.Word("␣"))) + pp.CaselessLiteral("repeat")
137 TK_END_REPEAT = (pp.lineStart() ^ pp.OneOrMore(pp.Word("␣"))) +
138                 pp.CaselessLiteral("end_repeat").setParseAction( pp.replaceWith("END_WHILE"))
139 TK_UNTIL = (pp.lineStart() | pp.OneOrMore(pp.Word("␣"))) +
140            pp.CaselessLiteral("until␣").setParseAction(pp.replaceWith("\nwhile␣not("))
141 SQ_REPEAT = TK_REPEAT + pp.OneOrMore(SQ_SET_VALUE_HELPER_CASE)
142 SQ_UNTIL = TK_UNTIL + SQ_RELATION +
143            pp.Optional(pp.OneOrMore((pp.Word("and␣AND␣&").setParseAction(pp.replaceWith("and")) |
144            pp.CaselessLiteral("or") | pp.CaselessLiteral("xor")) + SQ_RELATION))
145 TK_FLAG_FOR = pp.Literal("flag_for").setParseAction(pp.replaceWith(""))
```

```
    SQ_REAL_TO_WORD = TK_VARIABLE + pp.Optional(pp.OneOrMore(pp.Word("␣"))) + TK_EQUAL +
                      pp.Optional(pp.OneOrMore(pp.Word("␣"))) + pp.CaselessLiteral("REAL_TO_WORD") +
                      TK_VARIABLE + TK_SEMICOLON
    TK_TYPE_CONVERTING_INT = (pp.Word(pp.alphas) + pp.Word("_") + pp.CaselessLiteral("to") +
                              pp.Word("_") +
                              pp.CaselessKeyword("INT")).setParseAction(pp.replaceWith("INT")) +
                              TK_VARIABLE
    TK_TYPE_CONVERTING_REAL = (pp.Word(pp.alphas) + pp.Word("_") + pp.CaselessLiteral("to") +
                               pp.Word("_") +
                               pp.CaselessKeyword("REAL")).setParseAction(pp.replaceWith("float")) +
                               TK_VARIABLE
    TK_TYPE_CONVERTING_OTHERS = (pp.Word(pp.alphas) + pp.Word("_") + pp.CaselessLiteral("to") +
                                 pp.Word("_") +
                                 pp.Word(pp.alphas)).setParseAction(pp.replaceWith("")) + TK_VARIABLE
    SQ_SEL = (pp.lineStart() | pp.OneOrMore(pp.Word("␣"))) + TK_VARIABLE + TK_EQUAL + \
             pp.CaselessLiteral("SEL") + pp.Literal("(") + pp.Word(pp.alphanums + "_" + "-") +
             TK_COMMA + pp.Word(pp.alphanums + "_" + "-" + "+" + "␣") + TK_COMMA +
             pp.Word(pp.alphanums + "_" + "-" + "+" + "␣") + pp.Literal(")")
    SQ_LIMIT = (pp.lineStart() | pp.OneOrMore(pp.Word("␣"))) + TK_VARIABLE + TK_EQUAL + \
               pp.CaselessLiteral("LIMIT") + pp.Literal("(") + pp.Word(pp.alphanums) + TK_COMMA + \
               pp.Word(pp.alphanums) + TK_COMMA + pp.Word(pp.alphanums) + pp.Literal(")")
    SQ_MUX = (pp.lineStart() | pp.OneOrMore(pp.Word("␣"))) + TK_VARIABLE + TK_EQUAL + \
        pp.CaselessLiteral("MUX") + pp.Literal("(") + pp.Word(pp.alphanums) + TK_COMMA + \
        pp.Word(pp.alphanums) + pp.OneOrMore(TK_COMMA + pp.Word(pp.alphanums)) + pp.Literal(")")
    TK_DATA_TYPE = pp.Word(pp.alphanums + "_" + "-") + TK_COLON +
                   pp.oneOf("BYTE␣WORD␣DWORD␣INT␣BOOL␣REAL␣TIME␣TON␣TOF␣TP␣PID")
    TK_DATA_TYPE_DELETE = (pp.Word(pp.alphanums + "_" + "-") + TK_COLON +
                           pp.oneOf("BYTE␣WORD␣DWORD␣INT␣BOOL␣REAL␣TIME␣TON␣TOF␣TP")).\
                           setParseAction(pp.replaceWith(""))
    TK_CONST = pp.OneOrMore(pp.Word("␣")) + pp.Word(pp.alphanums + "_" + "-") + TK_COLON +
               (((pp.oneOf("BYTE␣WORD␣DWORD␣INT␣BOOL␣REAL␣TIME") + pp.Optional(TK_EQUAL +
               pp.Word(pp.alphanums + "." + "-")))) | (pp.Literal("ARRAY[0..") + pp.Word(pp.nums) +
               pp.Literal("]␣OF␣") + pp.oneOf("BYTE␣WORD␣DWORD␣INT␣BOOL␣REAL␣TIME"))) + TK_SEMICOLON
    TK_VAR_DEF = pp.Word(pp.alphanums + "_" + "-") + TK_COLON + pp.oneOf("BYTE␣WORD␣DWORD␣INT␣BOOL\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣REAL␣TIME␣TON␣TOF␣TP") + \
        pp.ZeroOrMore(TK_EQUAL + pp.Word(pp.alphanums + ".")) + TK_SEMICOLON
    TK_CONST_DELETE = (pp.Word(pp.alphanums + "_" + "-") + TK_COLON +
                       pp.oneOf("BYTE␣WORD␣DWORD␣INT␣BOOL␣REAL␣TIME␣TON␣TOF␣TP")  TK_EQUAL +
                       pp.Word(pp.nums)).setParseAction(pp.replaceWith(""))
    TK_TIMER = pp.oneOf("TON␣TOF␣TP")
    TK_TIMER_DEFINITION = TK_VARIABLE + pp.CaselessLiteral("(in") + TK_EQUAL +
                          pp.Word(pp.alphanums + "_") +  pp.Optional((pp.CaselessLiteral(",␣pt") |
                          pp.CaselessLiteral(",pt")) + TK_EQUAL + TK_TIME) + TK_SEMICOLON
    TK_TIMER_Q_ET = pp.Word(pp.alphanums + "_") + (pp.Literal(".Q") | pp.Literal(".q") |
                    pp.Literal(".ET") | pp.Literal(".et"))
```

Quellcode A.3: Token/Sequenzen des CODESYSImporters

```
    SQ_START_PID = pp.Literal("_END_COMMENT\n_ASSIGN\n_FUNCTIONBLOCK")
    TK_BOX_EXPR = pp.Literal("_BOX_EXPR␣:")
    SQ_EXPRESSION_PID = pp.Literal("_OPERAND\n_EXPRESSION\n_POSITIV")
    SQ_EXPRESSION_PID_2 = pp.Literal("_EXPRESSION\n_POSITIV\nPID\n_OUTPUTS␣:")
    SQ_EXPRESSION_PID_3 = pp.Literal("_OUTPUT\n_POSITIV\n_NO_SET")
    SQ_EXPRESSION_PID_4 = pp.Literal("_EXPRESSION\n_POSITIV\n_OUTPUTS␣:␣\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣1\n_OUTPUT\n_POSITIV\n_NO_SET")
    SQ_PID = SQ_START_PID + STParser.TK_VARIABLE + TK_BOX_EXPR + STParser.TK_REAL +
                            pp.OneOrMore(SQ_EXPRESSION_PID + STParser.TK_VARIABLE) + \
        SQ_EXPRESSION_PID_2 + STParser.TK_REAL + pp.OneOrMore(
        SQ_EXPRESSION_PID_3 + STParser.TK_VARIABLE) + SQ_EXPRESSION_PID_4 + STParser.TK_VARIABLE
    SQ_TASK_HEADER = pp.Literal("TASK") + STParser.TK_VARIABLE + pp.Literal("(PRIORITY␣:=") +
```

```
                    STParser.TK_REAL + pp.Optional(pp.Word(",")) + \
        pp.Optional(pp.Literal("␣SINGLE␣:=") + STParser.TK_VARIABLE + pp.Word(",")) + pp.Optional(
        pp.Literal("␣INTERVAL␣:=") + STParser.TK_TIME) + \
        pp.Optional(pp.Word(")")) + STParser.TK_SEMICOLON
    SQ_TASK = SQ_TASK_HEADER + pp.OneOrMore(STParser.TK_VARIABLE + pp.Literal("();\n")) +
        pp.Literal("{Additional_info␣:␣") + pp.OneOrMore(STParser.TK_REAL + pp.Word(",")) +
        pp.Word(pp.nums) + pp.Literal("}\nEND_TASK")
    TK_PROGRAM = pp.Group((pp.Literal("\nPROGRAM")|("\n␣␣PROGRAM")) + STParser.TK_VARIABLE)
    TK_PROGRAM_END = pp.Literal("\nEND_PROGRAM\n\n")
    SQ_HEADER_PID = (pp.Literal("\nPROGRAM") | pp.Literal("\n␣PROGRAM") | pp.Literal("\n␣␣PROGRAM")) +
        STParser.TK_VARIABLE + pp.Literal("VAR") + pp.OneOrMore(
        pp.ZeroOrMore(pp.White()) + STParser.TK_DATA_TYPE + pp.Literal(";")) +
        pp.Literal("\nEND_VAR")
    TK_PLC_CONFIG = pp.Literal("\nPLC_CONFIGURATION")
    TK_PLC_CONFIG_END = pp.Literal("\nPLC_END")
    TK_STRING = pp.quotedString | pp.sglQuotedString
    SQ_MODULE_CONFIG = pp.Optional(pp.Literal("\n_MODULE:")) + TK_STRING +
        pp.Literal("\n_SECTION_NAME:") + TK_STRING + pp.Literal("\n_INDEX_IN_PARENT:") + TK_STRING +
        pp.Literal("\n_MODULE_NAME:") + TK_STRING + pp.Literal("\n_NODE_ID:") + STParser.TK_REAL + \
        pp.Literal("_IECIN:␣") + pp.Word(pp.alphanums + "%" + ".") + pp.Literal("\n_IECOUT:␣") +
        pp.Word(pp.alphanums + "%" + ".") + pp.Literal("\n_IECDIAG:␣") + pp.Word(pp.alphanums + "%" +
        ".") + pp.Literal("\n_DOWNLOAD:␣") + pp.Word(pp.nums) + pp.Literal("\n_EXCLUDEFROMAUTOADR:␣")
        + (pp.Word(pp.nums) | pp.Word(pp.alphanums + "%" + ".")) + pp.Literal("\n_COMMENT:") +
        TK_STRING
    TK_END_MODULE = pp.Literal("\n_END_MODULE") | pp.Literal("\n\n_MODULE:")
    SQ_HW_CONFIG = SQ_MODULE_CONFIG + TK_END_MODULE
    SQ_PLC_PARAM = pp.Literal("\n\n_PARAMETER\n_PARAM␣10000:␣") + pp.Word(pp.nums) + pp.Literal(",")
        + TK_STRING + pp.Literal("\n_PARAM␣10001:␣") + pp.Word(pp.nums) + pp.Literal(",") +
        TK_STRING + pp.Literal("\n_END_PARAMETER")
    SQ_PLC_CHANNEL = pp.Literal("\n\n_CHANNEL") + pp.Literal("\n_SECTION_NAME:") + TK_STRING +
        pp.Literal("\n_INDEX_IN_PARENT:") + TK_STRING + pp.Literal("\n_SYMBOLIC_NAME:") + TK_STRING +
        pp.Literal("\n_COMMENT:") + TK_STRING + pp.Literal("\n_CHANNEL_MODE:") + TK_STRING + \
        pp.Literal("\n_IECADR:␣") + pp.Word(pp.alphanums + "%" + ".") + pp.Literal("\n_END_CHANNEL")
    SQ_HW_CONFIG_CLAMP = SQ_MODULE_CONFIG + SQ_PLC_PARAM + pp.OneOrMore(SQ_PLC_CHANNEL)
    TK_GLOBAL_VARS_START = pp.Literal("\nVAR_GLOBAL")
    TK_VARS_END = pp.Literal("\nEND_VAR")
```

Quellcode A.4: CODESYS Export-Datei des Steuerungsprogramm der iWÜST (inklusive Anwenderprogramme, teilweise Umbrüche nachträglich eingefügt

```
1
2
3  (* @NESTEDCOMMENTS := 'Yes' *)
4  (* @PATH := '\/Configuration' *)
5  (* @OBJECTFLAGS := '0, 8' *)
6  (* @SYMFILEFLAGS := '2048' *)
7  PROGRAM MODBUS_CONFIGURATION
8  VAR
9  END_VAR
10 (* @END_DECLARATION := '0' *)
11 (*
12    <?xml version="1.0" encoding="UTF-16" standalone="yes"?>
13 <network xml_structure_version="1" xml_feature_version="1">
14         <generator_settings minRTUTaskCycleTimeMs="5" minEthaskCycleTimeMs="5"
15          ethSlaveMultiplier="0.1" rtuInterfaceMultiplier="0.1" taskGeneration="false"/>
16 </network>
17
18 *)
```

```
19 ;
20 END_PROGRAM
21
22
23 (* @NESTEDCOMMENTS := 'Yes' *)
24 (* @PATH := '\/Configuration' *)
25 (* @OBJECTFLAGS := '0, 8' *)
26 (* @SYMFILEFLAGS := '2048' *)
27 PROGRAM MODBUS_CONFIGURATION_1
28 VAR
29 END_VAR
30 (* @END_DECLARATION := '0' *)
31 (*
32    <?xml version="1.0" encoding="UTF-16" standalone="yes"?>
33 <network xml_structure_version="1" xml_feature_version="1">
34         <generator_settings minRTUTaskCycleTimeMs="5" minEthaskCycleTimeMs="5"
35          ethSlaveMultiplier="0.1" rtuInterfaceMultiplier="0.1" taskGeneration="false"/>
36 </network>
37
38 *)
39 ;
40 END_PROGRAM
41
42
43 (* @NESTEDCOMMENTS := 'Yes' *)
44 (* @PATH := '' *)
45 (* @OBJECTFLAGS := '0, 8' *)
46 (* @SYMFILEFLAGS := '2048' *)
47 PROGRAM PID_generator
48 VAR
49         pid_generator: PID;
50 END_VAR
51 (* @END_DECLARATION := '0' *)
52 _FBD_BODY
53 _NETWORKS : 1
54 _NETWORK
55
56 _COMMENT
57 ''
58 _END_COMMENT
59 _ASSIGN
60 _FUNCTIONBLOCK
61 pid_generator
62 _BOX_EXPR : 11
63 _OPERAND
64 _EXPRESSION
65 _POSITIV
66 T_Netz_ist
67 _OPERAND
68 _EXPRESSION
69 _POSITIV
70 T_Netz_soll
71 _OPERAND
72 _EXPRESSION
73 _POSITIV
74 0.00005
75 _OPERAND
76 _EXPRESSION
77 _POSITIV
78 10000000
```

```
79  _OPERAND
80  _EXPRESSION
81  _POSITIV
82  0
83  _OPERAND
84  _EXPRESSION
85  _POSITIV
86  0
87  _OPERAND
88  _EXPRESSION
89  _POSITIV
90  0
91  _OPERAND
92  _EXPRESSION
93  _POSITIV
94  -0.02
95  _OPERAND
96  _EXPRESSION
97  _POSITIV
98  0.02
99  _OPERAND
100 _EXPRESSION
101 _POSITIV
102 FALSE
103 _OPERAND
104 _EXPRESSION
105 _POSITIV
106 FALSE
107 _EXPRESSION
108 _POSITIV
109 PID
110 _OUTPUTS : 2
111 _OUTPUT
112 _POSITIV
113 _NO_SET
114 _EMPTY
115 _OUTPUT
116 _POSITIV
117 _NO_SET
118 _EMPTY
119 _EXPRESSION
120 _POSITIV
121 _OUTPUTS : 1
122 _OUTPUT
123 _POSITIV
124 _NO_SET
125 generator_clamp_
126
127 END_PROGRAM
128
129
130 (* @NESTEDCOMMENTS := 'Yes' *)
131 (* @PATH := '' *)
132 (* @OBJECTFLAGS := '0, 8' *)
133 (* @SYMFILEFLAGS := '2048' *)
134   PROGRAM PID_MVdreinulleins
135 VAR
136         pid_MV301: PID;
137 END_VAR
138 (* @END_DECLARATION := '0' *)
```

```
139 _FBD_BODY
140 _NETWORKS : 1
141 _NETWORK
142
143 _COMMENT
144 ' '
145 _END_COMMENT
146 _ASSIGN
147 _FUNCTIONBLOCK
148 pid_MV301
149 _BOX_EXPR : 11
150 _OPERAND
151 _EXPRESSION
152 _POSITIV
153 MV301_ist
154 _OPERAND
155 _EXPRESSION
156 _POSITIV
157 MV301_soll
158 _OPERAND
159 _EXPRESSION
160 _POSITIV
161 500
162 _OPERAND
163 _EXPRESSION
164 _POSITIV
165 30
166 _OPERAND
167 _EXPRESSION
168 _POSITIV
169 0
170 _OPERAND
171 _EXPRESSION
172 _POSITIV
173 0
174 _OPERAND
175 _EXPRESSION
176 _POSITIV
177 0
178 _OPERAND
179 _EXPRESSION
180 _POSITIV
181 0
182 _OPERAND
183 _EXPRESSION
184 _POSITIV
185 32767
186 _OPERAND
187 _EXPRESSION
188 _POSITIV
189 FALSE
190 _OPERAND
191 _EXPRESSION
192 _POSITIV
193 FALSE
194 _EXPRESSION
195 _POSITIV
196 PID
197 _OUTPUTS : 2
198 _OUTPUT
```

```
199 _POSITIV
200 _NO_SET
201 _EMPTY
202 _OUTPUT
203 _POSITIV
204 _NO_SET
205 _EMPTY
206 _EXPRESSION
207 _POSITIV
208 _OUTPUTS : 1
209 _OUTPUT
210 _POSITIV
211 _NO_SET
212 MV301_clamp_
213
214 END_PROGRAM
215
216
217 (* @NESTEDCOMMENTS := 'Yes' *)
218 (* @PATH := '' *)
219 (* @OBJECTFLAGS := '0, 8' *)
220 (* @SYMFILEFLAGS := '2048' *)
221 PROGRAM PID_MVeinsnulleins
222 VAR
223         pid_MV101: PID;
224 END_VAR
225 (* @END_DECLARATION := '0' *)
226 _FBD_BODY
227 _NETWORKS : 1
228 _NETWORK
229
230 _COMMENT
231 ''
232 _END_COMMENT
233 _ASSIGN
234 _FUNCTIONBLOCK
235 pid_MV101
236 _BOX_EXPR : 11
237 _OPERAND
238 _EXPRESSION
239 _POSITIV
240 MV101_ist
241 _OPERAND
242 _EXPRESSION
243 _POSITIV
244 MV101_soll
245 _OPERAND
246 _EXPRESSION
247 _POSITIV
248 Kp_MV101
249 _OPERAND
250 _EXPRESSION
251 _POSITIV
252 Tn_MV101
253 _OPERAND
254 _EXPRESSION
255 _POSITIV
256 Tv_MV101
257 _OPERAND
258 _EXPRESSION
```

```
259 _POSITIV
260 0
261 _OPERAND
262 _EXPRESSION
263 _POSITIV
264 0
265 _OPERAND
266 _EXPRESSION
267 _POSITIV
268 0
269 _OPERAND
270 _EXPRESSION
271 _POSITIV
272 32767
273 _OPERAND
274 _EXPRESSION
275 _POSITIV
276 FALSE
277 _OPERAND
278 _EXPRESSION
279 _POSITIV
280 FALSE
281 _EXPRESSION
282 _POSITIV
283 PID
284 _OUTPUTS : 2
285 _OUTPUT
286 _POSITIV
287 _NO_SET
288 _EMPTY
289 _OUTPUT
290 _POSITIV
291 _NO_SET
292 _EMPTY
293 _EXPRESSION
294 _POSITIV
295 _OUTPUTS : 1
296 _OUTPUT
297 _POSITIV
298 _NO_SET
299 MV101_clamp_
300
301 END_PROGRAM
302
303
304 (* @NESTEDCOMMENTS := 'Yes' *)
305 (* @PATH := '' *)
306 (* @OBJECTFLAGS := '0, 8' *)
307 (* @SYMFILEFLAGS := '2048' *)
308 PROGRAM PID_Pviernulleins
309 VAR
310         pid_P401: PID;
311 END_VAR
312 (* @END_DECLARATION := '0' *)
313 _FBD_BODY
314 _NETWORKS : 1
315 _NETWORK
316
317 _COMMENT
318 ''
```

```
319 _END_COMMENT
320 _ASSIGN
321 _FUNCTIONBLOCK
322 pid_P401
323 _BOX_EXPR : 11
324 _OPERAND
325 _EXPRESSION
326 _POSITIV
327 TIRC401_clamp_
328 _OPERAND
329 _EXPRESSION
330 _POSITIV
331 TIRC401_clamp_soll
332 _OPERAND
333 _EXPRESSION
334 _POSITIV
335 0.00005
336 _OPERAND
337 _EXPRESSION
338 _POSITIV
339 10000000
340 _OPERAND
341 _EXPRESSION
342 _POSITIV
343 0
344 _OPERAND
345 _EXPRESSION
346 _POSITIV
347 0
348 _OPERAND
349 _EXPRESSION
350 _POSITIV
351 0
352 _OPERAND
353 _EXPRESSION
354 _POSITIV
355 −0.02
356 _OPERAND
357 _EXPRESSION
358 _POSITIV
359 0.02
360 _OPERAND
361 _EXPRESSION
362 _POSITIV
363 FALSE
364 _OPERAND
365 _EXPRESSION
366 _POSITIV
367 FALSE
368 _EXPRESSION
369 _POSITIV
370 PID
371 _OUTPUTS : 2
372 _OUTPUT
373 _POSITIV
374 _NO_SET
375 _EMPTY
376 _OUTPUT
377 _POSITIV
378 _NO_SET
```

```
379 _EMPTY
380 _EXPRESSION
381 _POSITIV
382 _OUTPUTS : 1
383 _OUTPUT
384 _POSITIV
385 _NO_SET
386 P401_clamp_
387
388 END_PROGRAM
389
390
391 (* @NESTEDCOMMENTS := 'Yes' *)
392 (* @PATH := '' *)
393 (* @OBJECTFLAGS := '0, 8' *)
394 (* @SYMFILEFLAGS := '2048' *)
395 PROGRAM PLC_PRG_TWW
396 VAR
397 (*Temperaturen*)
398         TIRC324_clamp_: REAL;
399         TIRC323_clamp_: REAL;
400         TIRC321_clamp_: REAL;
401         TIRC320_clamp_: REAL;
402         TIR301_clamp_: REAL;
403         TIRC305_clamp_: REAL;
404         TIRC103_clamp_: REAL;
405         temp_sensor_help_clamp: WORD;
406         FLAG_SET_OFFSETS_TWW: BOOL := FALSE;
407 (*Berechnung SOC*)
408         T_med: REAL;
409         T_RL_min: REAL := 25;
410         T_Netz_ARRAY: ARRAY[0..60] OF REAL;
411         T_Netz_sum: REAL;
412         Zaehler_Netz: INT := 0;
413         Zaehler_Netz_2: INT := 0;
414         SOC_Leitwarte: REAL := 80;
415         SOC_min: REAL := 33;
416         SOC_max: REAL := 100;
417         SOC_soll_min: REAL;
418 (*Berechnung T_Netz*)
419         T_delta: REAL;
420         T_aussen: REAL := 31;
421         T_Netz_start: REAL := 85;
422         T_aussen_max: REAL := 30;
423         T_aussen_min: REAL := -10;
424 (*Variablen_RL_Begrenzung*)
425         TIRC302_clamp_max: REAL := 55;
426         TIRC302_clamp_zulaessig: REAL := 50;
427         SOC_RL_Begrenzung: REAL := 10;
428         FLAG_9_TWW: BOOL;
429 (*Variablen Leistungsbegrenzung*)
430         Timer_WMZ: TON;
431         P_WMZ001_ARRAY15min: ARRAY[0..29] OF REAL;
432         Timer_WMZ_IN_TWW: BOOL := TRUE;
433         Timer_WMZ_Q_TWW: BOOL;
434         P_WMZ001_15min: REAL;
435         P_ARRAY: INT;
436         Zaehler : INT;
437         P_WMZ001_15min_mittel: REAL;
438         FLAG_8_TWW: BOOL;
```

```
439 (*Variablen Fuellstandsbegrenzung*)
440         T_delta_Fuellstandsbegrenzung_in: REAL := 10;
441         T_delta_Fuellstandsbegrenzung_out:REAL := 15;
442         SOC_Fuellstandsbegrenzung: REAL := 50;
443         T_delta_VL_RL: REAL;
444         FLAG_7_TWW: BOOL;
445         Timer_Fuellstandsbegrenzung: TOF;
446         Zaehler_Fuellstandsbegrenzung: INT := 0;
447 (*Variablen Speicher ueberladen*)
448         EXTERN_Fuellstand_max_aktiv: BOOL;
449         FLAG_6_TWW: BOOL;
450         Timer_Speicherueberladen: TOF;
451         Fuellstandsbegrenzung_deaktiviert: BOOL;
452 (*Hysterese Timer SOC*)
453         Timer_SOC_2min: BOOL;
454         SOC_Timer : TOF;
455         SOC_Timer_q: BOOL;
456 (*Funktionspruefung*)
457         SOC_freigabe_funktionspruefung: REAL := 20;
458         EXTERN_DO_funktionspruefung_TWW: BOOL;
459         start_funktionspruefung_5_TWW: BOOL;
460         start_funktionspruefung_TWW: BOOL;
461         READY_funktionspruefung: BOOL;
462         FLAG_5_TWW: BOOL;
463         Massenstrom_ges_TWW: REAL;
464         Massenstrom_max_TWW: REAL := 2.4;
465         Massenstrom_min_TWW: REAL := 0;
466         Zaehler_5_TWW: INT := 0;
467         (*Ergebnis_funktionspruefung_TWW: WORD;*)
468 (*Netzkollaps*)
469         FLAG_4_TWW: BOOL;
470 (*Kommunikationsausfall*)
471         Timer_Toggle: TON;
472         TOGGLE: BOOL;
473         IN_Timer_Toggle: BOOL;
474 (*Minimale Temperatur*)
475         TIRC324_clamp_min_hT: REAL := 70;
476         TIRC324_clamp_min_nT: REAL := 65;
477         TIRC324_clamp_min: REAL;
478         TIRC324_clamp_soll_speicher_kopfraum_hT: REAL := 76;
479         TIRC324_clamp_soll_speicher_kopfraum_nT: REAL := 70;
480         TIRC324_clamp_soll_speicher_kopfraum: REAL;
481         TIRC324_clamp_ladung_ausstieg_hT: REAL := 74;
482         TIRC324_clamp_ladung_ausstieg_nT: REAL := 68;
483         TIRC324_clamp_ladung_ausstieg: REAL;
484         TIRC32X_clamp_grenz: REAL;
485         Storage_min_temp_in: BOOL := FALSE;
486         Storage_min_temp_out: BOOL := FALSE;
487         TIRC324_clamp_soll: REAL;
488         T_Netz_ausstieg: REAL;
489         T_Netz_gering: REAL := 75;
490 (*Inbetriebnahme*)
491         EXTERN_Inbetriebnahme_start_TWW: BOOL;
492         EXTERN_Inbetriebnahme_abgeschlossen_TWW: BOOL;
493         Inbetriebnahme_Szenario_TWW: INT;
494         Inbetriebnahme_Temps_TWW: INT;
495         STB_pruef_TWW: BOOL;
496         FLAG_INBE_2_TWW: BOOL;
497         FLAG_INBE_5_TWW: BOOL;
498         Temp_min: REAL;
```

```
499         Temp_max: REAL;
500 (*Waechter*)
501         (*FLAG_0_TWW: BOOL;*)
502 END_VAR
503 (* @END_DECLARATION := '0' *)
504 IF FLAG_JARVIS = TRUE THEN
505 OFFSET_Kelvin := 273.15;
506 Skalierung_Temp_clamp := 1;
507 END_IF;
508 IF FLAG_JARVIS = FALSE THEN
509 OFFSET_Kelvin := 0;
510 Skalierung_Temp_clamp := 10;
511 END_IF;
512
513 (*Einlesen der Temperaturen an den Klemmen und in Grad Celsius umrechnen*)
514 TIRC324_clamp_ := WORD_TO_REAL(storage1_TIRC324 )/ Skalierung_Temp_clamp;
515 TIRC323_clamp_ := WORD_TO_REAL(storage1_TIRC323) / Skalierung_Temp_clamp;
516 TIRC322_clamp_ := WORD_TO_REAL(storage1_TIRC322) / Skalierung_Temp_clamp;
517 TIRC321_clamp_ := WORD_TO_REAL(storage2_TIRC321) / Skalierung_Temp_clamp;
518 TIRC320_clamp_ := WORD_TO_REAL(storage2_TIRC320) / Skalierung_Temp_clamp;
519 TIRC401_clamp_ := WORD_TO_REAL(TIRC401_clamp) / Skalierung_Temp_clamp;
520 TIR301_clamp_ := WORD_TO_REAL(TIR301_clamp) / Skalierung_Temp_clamp;
521 TIRC302_clamp_ := WORD_TO_REAL(TIRC302_clamp) / Skalierung_Temp_clamp;
522 TIRC305_clamp_ := WORD_TO_REAL(TIRC305_clamp) / Skalierung_Temp_clamp;
523 TIRC103_clamp_ := WORD_TO_REAL(TIRC103_clamp) / Skalierung_Temp_clamp;
524
525 IF FLAG_JARVIS = TRUE THEN
526 T_Netz_ist := WORD_TO_REAL(temp_sensor_help_clamp) / Skalierung_Temp_clamp;
527 END_IF;
528 (*Skalierung der Konstanten - Kelvin/Grad Celsius, wird nur beim ersten Programmaufruf durchgeführt*)
529 IF FLAG_SET_OFFSETS_TWW = FALSE THEN
530 T_RL_min := T_RL_min + OFFSET_Kelvin;
531 T_Netz_start := T_Netz_start + OFFSET_Kelvin;
532 T_aussen := T_aussen + OFFSET_Kelvin;
533 T_aussen_max := T_aussen_max + OFFSET_Kelvin;
534 T_aussen_min := T_aussen_min + OFFSET_Kelvin;
535 TIRC302_clamp_max := TIRC302_clamp_max + OFFSET_Kelvin;
536 TIRC302_clamp_zulaessig := TIRC302_clamp_zulaessig + OFFSET_Kelvin;
537 TIRC324_clamp_min_hT := TIRC324_clamp_min_hT + OFFSET_Kelvin;
538 TIRC324_clamp_min_nT := TIRC324_clamp_min_nT + OFFSET_Kelvin;
539 TIRC324_clamp_soll_speicher_kopfraum_hT := TIRC324_clamp_soll_speicher_kopfraum_hT + OFFSET_Kelvin;
540 TIRC324_clamp_soll_speicher_kopfraum_nT := TIRC324_clamp_soll_speicher_kopfraum_nT + OFFSET_Kelvin;
541 TIRC324_clamp_ladung_ausstieg_hT := TIRC324_clamp_ladung_ausstieg_hT + OFFSET_Kelvin;
542 TIRC324_clamp_ladung_ausstieg_nT := TIRC324_clamp_ladung_ausstieg_nT + OFFSET_Kelvin;
543 TIRC401_clamp_soll := TIRC401_clamp_soll + OFFSET_Kelvin;
544 TIRC_max := TIRC_max + OFFSET_Kelvin;
545 TIRC_min := TIRC_min + OFFSET_Kelvin;
546 T_Netz_gering := T_Netz_gering + OFFSET_Kelvin;
547 T_Netz_soll := T_Netz_soll + OFFSET_Kelvin;
548 END_IF;
549 FLAG_SET_OFFSETS_TWW := TRUE;
550
551 (*-------Berechnung Aussentemperatur & Netztemperatur*)
552 IF T_aussen > T_aussen_max THEN
553 T_delta := -0.1;
554 ELSIF T_aussen < T_aussen_min THEN
555 T_delta := 0.1;
556 END_IF;
557
558 (*Fuer langsames Schwanken der Netztemperatur untere Zeile einfuegen, T_aussen verändert sich in Code
```

```
559 vom WMZ*)
560 (*T_Netz_soll := T_Netz_start - ((T_aussen-273)/2);*)
561
562 IF FLAG_JARVIS = FALSE THEN
563 T_Netz_soll := T_Netz_start - (T_aussen/2);
564 T_Netz_ist := T_Netz_soll;
565 END_IF;
566
567 T_med := (TIRC320_clamp_ + TIRC321_clamp_ + TIRC322_clamp_ + TIRC323_clamp_ + TIRC324_clamp_)/5;
568 (*Berechnung Mittelwert Netztemp fuer letzte Minute in Simulation*)
569 Zaehler_Netz := Zaehler_Netz + 1;
570 T_Netz_ARRAY[Zaehler_Netz] := T_Netz_ist;
571 IF Zaehler_Netz = 60 THEN
572 T_Netz_sum := 0;
573 Zaehler_Netz := 0;
574 Zaehler_Netz_2 := 0;
575 END_IF;
576 WHILE Zaehler_Netz_2 < 61 DO
577 T_Netz_sum := T_Netz_sum + T_Netz_ARRAY[Zaehler_Netz_2];
578 Zaehler_Netz_2 := Zaehler_Netz_2 + 1;
579 END_WHILE;
580 T_Netz_mittel := T_Netz_sum/60;
581
582 SOC := (T_med - T_RL_min)/(T_Netz_mittel - T_RL_min)*100;
583
584 (*----------Normalbetrieb----------*)
585
586 Szenario_TWW := 10;
587 SOC_soll := SOC_Leitwarte;
588
589 (*----------RL-Begrenzung----------*)
590 IF Zaehler_gesamt > 61 THEN
591 IF TIRC302_clamp_ > TIRC302_clamp_max OR FLAG_9_TWW = TRUE THEN
592 Szenario_TWW := 9;
593 SOC_soll := SOC_RL_Begrenzung;
594 FLAG_9_TWW := TRUE;
595 END_IF;
596 IF TIRC302_clamp_ < TIRC302_clamp_zulaessig THEN
597 FLAG_9_TWW := FALSE;
598 END_IF;
599
600 (*----------Leistungsbegrenzung----------*)
601 (*Annahme: Oeffnung Mischventil entspricht Durchfluss in l/s, in realem System kommt P von WMZ*)
602 P_WMZ001 := (( ((MV101_clamp_ - 0.004)/0.016)*0.7*(T_Netz_mittel - TIRC103_clamp_) + ((MV301_clamp_ -
603             0.004)/0.016)*1.7*(T_Netz_mittel - TIRC305_clamp_))*c_p)/1000; (*[kW]*)
604 IF FLAG_JARVIS = FALSE THEN
605 P_WMZ001 := (((MV101_clamp_/32767)*0.7*(T_Netz_mittel - TIRC103_clamp_) + (MV301_clamp_/32767)*1.7*
606              (T_Netz_mittel - TIRC305_clamp_))*c_p)/1000; (*[kW]*)
607 END_IF;
608
609 (*Simulierter WMZ liest Leistung alle 30 Sekunden aus:*)
610 Timer_WMZ(IN:=Timer_WMZ_IN_TWW, PT := T#30s);
611 Timer_WMZ_Q_TWW := Timer_WMZ.Q;
612 Timer_WMZ_IN_TWW := TRUE;
613 IF Timer_WMZ_Q_TWW = TRUE THEN
614 Timer_WMZ_IN_TWW := FALSE;
615 (*Aussentemperatur simulieren*)
616 T_aussen := T_aussen + T_delta;
617 (*Speichern der Leistung in Array*)
618 P_WMZ001_ARRAY15min[P_ARRAY] := P_WMZ001;
```

```
619 P_ARRAY := P_ARRAY + 1;
620 Zaehler := 0;
621 P_WMZ001_15min := 0;
622 (*Berechnung Mittelwert Leistung 15min*)
623 WHILE Zaehler < 30  DO
624 P_WMZ001_15min := P_WMZ001_15min + P_WMZ001_ARRAY15min[Zaehler];
625 Zaehler := Zaehler + 1;
626 END_WHILE;
627 END_IF;
628
629 IF P_ARRAY = 30 THEN
630 P_ARRAY := 0;
631 END_IF;
632
633 P_WMZ001_15min_mittel := P_WMZ001_15min/30;
634 Grenzwert_kulanz := P_WMZ001_zulaessig * f_ueberschreitung_zulaessig;
635 (*Abfrage Leistungsbegrenzung*)
636 IF P_WMZ001_15min_mittel > Grenzwert_kulanz OR FLAG_8_TWW = TRUE THEN
637 FLAG_8_TWW := TRUE;
638 Szenario_TWW := 8;
639 END_IF;
640 IF P_WMZ001_15min_mittel < P_WMZ001_zulaessig THEN
641 FLAG_8_TWW := FALSE;
642 END_IF;
643
644 (*Aktivierung des Timers, damit SOC_soll nicht springt:*)
645 CASE Szenario_TWW OF
646 8: Timer_SOC_2min := TRUE;
647 ELSE Timer_SOC_2min := FALSE;
648 END_CASE;
649
650 (*Timer_SOC_2min haelt SOC_soll 2min auf SOC_min*)
651 SOC_Timer(IN := Timer_SOC_2min, PT := T#2m);
652 SOC_Timer_q := SOC_Timer.Q;
653 IF SOC_Timer_q = TRUE THEN
654 SOC_soll := SOC_min;
655 Szenario_TWW := 8;
656 END_IF;
657
658 (*----------Fuellstandsbegrenzung----------*)
659 (*Fuellstandsbegrenzung wird deaktiviert, falls Speicher ueberladen in den letzten 8 Minuten aktiv
660 war*)
661 IF Fuellstandsbegrenzung_deaktiviert = FALSE THEN
662
663 T_delta_VL_RL := T_Netz_mittel - TIRC305_clamp_;
664 (*Zaehler, damit Sprung in temp_sensor_help nicht beeinflusst*)
665 IF Zaehler_Fuellstandsbegrenzung = 0 OR Zaehler_Fuellstandsbegrenzung > 20 THEN
666 IF T_delta_VL_RL <= T_delta_Fuellstandsbegrenzung_in THEN
667 FLAG_7_TWW := TRUE;
668 END_IF;
669 IF T_delta_VL_RL >= T_delta_Fuellstandsbegrenzung_out THEN
670 FLAG_7_TWW := FALSE;
671 END_IF;
672 END_IF;
673
674 (*Timer, damit szenario 7min. fuer 8 minuten aktiv ist*)
675 Timer_Fuellstandsbegrenzung(IN := FLAG_7_TWW, PT := T#8m);
676 FLAG_Timer_Fuellstandsbegrenzung := Timer_Fuellstandsbegrenzung.Q;
677
678 IF FLAG_Timer_Fuellstandsbegrenzung = TRUE THEN
```

```
679 Szenario_TWW := 7;
680 SOC_soll := SOC_Fuellstandsbegrenzung;
681 Zaehler_Fuellstandsbegrenzung := Zaehler_Fuellstandsbegrenzung + 1;
682 ELSE Zaehler_Fuellstandsbegrenzung := 0;
683 END_IF;
684 END_IF;
685
686 (*-------------------------------Speicher ueberladen-------------------------------*)
687 IF EXTERN_Fuellstand_max_aktiv = TRUE THEN
688 Szenario_TWW := 6;
689 SOC_soll := SOC_max;
690 FLAG_6_TWW := TRUE; (*wird aktuell nicht benoetigt*)
691 ELSE FLAG_6_TWW := FALSE;
692 END_IF;
693
694 Timer_Speicherueberladen(IN := FLAG_6_TWW, PT := T#8m);
695 Fuellstandsbegrenzung_deaktiviert := Timer_Speicherueberladen.Q;
696
697 (*-------------------------------Funktionspruefung-------------------------------*)
698 (*Funktionspruefung kann durch EXTERN_DO_funktionspruefung und durch Inbetriebnahme aktiviert werden.
699 start_funktionspruefung_TWW verhindert Abbruch der Funktionspruefung, falls SOC waehrend Pruefung
700 unter 20% faellt*)
701 IF start_funktionspruefung_5_TWW = TRUE OR start_funktionspruefung_1_TWW = TRUE THEN
702 start_funktionspruefung_TWW := TRUE;
703 END_IF;
704 IF SOC > SOC_freigabe_funktionspruefung OR start_funktionspruefung_TWW = TRUE  THEN
705 READY_funktionspruefung := TRUE;
706 IF EXTERN_DO_funktionspruefung_TWW = TRUE  THEN
707 EXTERN_DO_funktionspruefung_WUE := TRUE;
708 start_funktionspruefung_5_TWW := TRUE;
709 Szenario_TWW := 5;
710 FLAG_5_TWW := TRUE;
711 ELSIF EXTERN_DO_funktionspruefung_TWW = FALSE AND EXTERN_Inbetriebnahme_start_TWW = FALSE THEN
712 IF EXTERN_Inbetriebnahme_start_WUE = FALSE THEN
713 start_funktionspruefung_5_TWW := FALSE;
714 start_funktionspruefung_TWW := FALSE;
715 FLAG_5_TWW := FALSE;
716 (*Ergebnis_funktionspruefung_TWW := 0;*)
717 (*STATUS_funktionspruefung_TWW := 0;*)
718 EXTERN_DO_funktionspruefung_WUE := FALSE;
719 END_IF;
720 END_IF;
721 END_IF;
722 IF start_funktionspruefung_5_TWW = TRUE OR start_funktionspruefung_1_TWW = TRUE THEN
723 (*In realem System wird Massenstrom durch WMZ geliefert*)
724 Massenstrom_ges_TWW := ((MV101_clamp_ - 0.004)/0.016)*0.7 + ((MV301_clamp_ - 0.004)/0.016)*1.7;
725 IF FLAG_JARVIS = FALSE THEN
726 Massenstrom_ges_TWW :=  (MV301_clamp_/32767)*1.7 + (MV101_clamp_/32767)*0.7;
727 END_IF;
728 (*Ventile schliesssen (via SETOUTPUT):*)
729 IF STATUS_funktionspruefung_TWW <> 2 AND STATUS_funktionspruefung_TWW <> 3 THEN
730 STATUS_funktionspruefung_TWW := 1;
731 END_IF;
732 IF STATUS_funktionspruefung_TWW = 1 THEN
733 (*Zaehler_5_TWW sorgt fuer Wartezeit von 30 Zeitschritten, damit Schalten sichtbar ist*)
734 Zaehler_5_TWW := Zaehler_5_TWW + 1;
735 (*Folgende 3 Zeilen vermindern Flackern, da PID-Regler  kurz  zwischenregeln*)
736 SOC_soll := SOC_min;
737 TIRC401_clamp_ := TIRC_max;
738 TIRC321_clamp_ := TIRC_max;
```

```
739 (*Ueberpruefung, ob minimaler Massenstrom gemessen wird*)
740 IF Massenstrom_ges_TWW = Massenstrom_min_TWW AND Zaehler_5_TWW > 30 THEN
741 STATUS_funktionspruefung_TWW := 2;
742 Zaehler_5_TWW := 0;
743 END_IF;
744 END_IF;
745 (*ventil oeffnen(via SETOUTPUT) : *)
746 IF STATUS_funktionspruefung_TWW = 2  THEN
747 Zaehler_5_TWW := Zaehler_5_TWW + 1;
748 (*Folgende 3 Zeilen vermindern Flackern, da PID-Regler  kurz  zwischenregeln*)
749 SOC_soll := SOC_max;
750 TIRC401_clamp_ := TIRC_min;
751 TIRC321_clamp_ := TIRC_min;
752 (*Ueberpruefung, ob maximaler Massenstrom gemessen wird*)
753 IF Massenstrom_ges_TWW = Massenstrom_max_TWW AND Zaehler_5_TWW > 30 THEN
754 STATUS_funktionspruefung_TWW := 3;
755 END_IF;
756 END_IF;
757 IF STATUS_funktionspruefung_TWW = 3 THEN
758 Ergebnis_funktionspruefung_TWW := 200;
759 (*Naechster Test der  Inbetriebnahme wird aktiviert*)
760 IF Inbetriebnahme_Szenario_TWW = 1 THEN
761 Inbetriebnahme_Szenario_TWW := 2;
762 END_IF;
763 STATUS_funktionspruefung_TWW := 0;
764 Zaehler_5_TWW := 0;
765 (*Schliesssen der Ventile gescheitert*)
766 ELSIF Zaehler_5_TWW = 500 AND STATUS_funktionspruefung_TWW = 1 THEN
767 Ergebnis_funktionspruefung_TWW := 40001;
768 STATUS_funktionspruefung_TWW := 1;
769 Zaehler_5_TWW := 0;
770 (*Oeffnen der Ventile gescheitert*)
771 ELSIF Zaehler_5_TWW = 500 AND STATUS_funktionspruefung_TWW = 2 THEN
772 Ergebnis_funktionspruefung_TWW := 40002;
773 STATUS_funktionspruefung_TWW := 1;
774 Zaehler_5_TWW := 0;
775 END_IF;
776 END_IF;
777
778 (*-----------------------------------Netzkollaps-----------------------------------*)
779 IF EXTERN_Netzkollaps = TRUE THEN
780 FLAG_4_TWW := TRUE;
781 Szenario_TWW := 4;
782 SOC_soll := 10;
783 ELSIF EXTERN_Netzkollaps = FALSE THEN
784 FLAG_4_TWW := FALSE;
785 END_IF;
786
787 (*---------------------------------Kommunikationsausfall---------------------------------*)
788 IF EXTERN_Toggle = TRUE AND TOGGLE = FALSE THEN
789 TOGGLE := TRUE;
790 IN_Timer_Toggle := FALSE;
791 ELSIF EXTERN_Toggle = FALSE AND TOGGLE = TRUE THEN
792 TOGGLE := FALSE;
793 IN_Timer_Toggle := FALSE;
794 ELSE IN_Timer_Toggle := TRUE;
795 END_IF;
796
797 Timer_Toggle(IN:=IN_Timer_Toggle, PT := T#15m);
798 FLAG_3 := Timer_Toggle.Q;
```

```
799 IF FLAG_3 = TRUE THEN
800 SOC_soll := 10;
801 Szenario_TWW := 3;
802 END_IF;
803
804 (*----------Minimale Temperatur----------*)
805 (*Anpassung der Grenzwerte, falls T_Netz gering*)
806 IF T_Netz_mittel <= T_Netz_gering THEN
807 TIRC324_clamp_min := TIRC324_clamp_min_nT;
808 TIRC324_clamp_soll_speicher_kopfraum := TIRC324_clamp_soll_speicher_kopfraum_nT;
809 TIRC324_clamp_ladung_ausstieg := TIRC324_clamp_ladung_ausstieg_nT;
810 ELSIF T_Netz_mittel > T_Netz_gering THEN
811 TIRC324_clamp_min := TIRC324_clamp_min_hT;
812 TIRC324_clamp_soll_speicher_kopfraum := TIRC324_clamp_soll_speicher_kopfraum_hT;
813 TIRC324_clamp_ladung_ausstieg := TIRC324_clamp_ladung_ausstieg_hT;
814 END_IF;
815 TIRC32X_clamp_grenz := TIRC401_clamp_soll + 3;
816 (*Ueberpruefung Eintrittskriterium*)
817 IF TIRC324_clamp_ < TIRC324_clamp_min AND TIRC323_clamp_ < TIRC32X_clamp_grenz THEN
818 Storage_min_temp_in := TRUE;
819 ELSE Storage_min_temp_in := FALSE;
820 END_IF;
821 IF Storage_min_temp_in = TRUE OR FLAG_2_TWW = TRUE THEN
822 Szenario_TWW := 2;
823 FLAG_2_TWW := TRUE;
824
825 TIRC324_clamp_soll := MIN(T_Netz_mittel, TIRC324_clamp_soll_speicher_kopfraum);
826 END_IF;
827 (*Ueberpruefen, ob Austrittskriterium erfuellt*)
828 T_Netz_ausstieg := T_Netz_mittel - 2;
829 IF TIRC324_clamp_ > TIRC324_clamp_ladung_ausstieg OR TIRC324_clamp_ > T_Netz_ausstieg THEN
830 Storage_min_temp_out := TRUE;
831 ELSE Storage_min_temp_out := FALSE;
832 END_IF;
833 IF Storage_min_temp_out = TRUE AND TIRC322_clamp_ > TIRC32X_clamp_grenz THEN
834 FLAG_2_TWW := FALSE;
835 END_IF;
836
837
838 (*----------Inbetriebnahme----------*)
839 IF EXTERN_Inbetriebnahme_start_TWW = TRUE AND EXTERN_Inbetriebnahme_abgeschlossen_TWW = FALSE THEN
840 (*Pingtest bestanden durch Empfangen des Bit EXTERN_Inbetriebnahme_start_TWW*)
841 Szenario_TWW := 1;
842 (*Start der Funktionspruefung*)
843 IF Ergebnis_funktionspruefung_TWW <> 200 THEN
844 Inbetriebnahme_Szenario_TWW := 1;
845 start_funktionspruefung_1_TWW := TRUE;
846 FLAG_INBE_2_TWW := TRUE;
847 END_IF;
848 (*Funktionspruefung bestanden, ueberpruefen der Messsignale*)
849 IF Ergebnis_funktionspruefung_TWW = 200 AND FLAG_INBE_2_TWW = TRUE THEN
850 Inbetriebnahme_Szenario_TWW := 2;
851 start_funktionspruefung_1_TWW := FALSE;
852 Temp_min := 0 + OFFSET_Kelvin;
853 Temp_max := 100 + OFFSET_Kelvin;
854 IF TIRC324_clamp_ < Temp_max AND TIRC324_clamp_ > Temp_min THEN
855 Inbetriebnahme_Temps_TWW := 1;
856 END_IF;
857 IF TIRC323_clamp_ < Temp_max AND TIRC323_clamp_ > Temp_min AND Inbetriebnahme_Temps_TWW = 1 THEN
858 Inbetriebnahme_Temps_TWW := 2;
```

```
859 END_IF;
860 IF TIRC322_clamp_ < Temp_max AND TIRC322_clamp_ > Temp_min AND Inbetriebnahme_Temps_TWW = 2 THEN
861 Inbetriebnahme_Temps_TWW := 3;
862 END_IF;
863 IF TIRC321_clamp_ < Temp_max AND TIRC321_clamp_ > Temp_min AND Inbetriebnahme_Temps_TWW = 3 THEN
864 Inbetriebnahme_Temps_TWW := 4;
865 END_IF;
866 IF TIRC320_clamp_ < Temp_max AND TIRC320_clamp_ > Temp_min AND Inbetriebnahme_Temps_TWW = 4 THEN
867 Inbetriebnahme_Temps_TWW := 5;
868 END_IF;
869 IF (*TIRC401_clamp_ < Temp_max AND TIRC401_clamp_ > Temp_min AND*) Inbetriebnahme_Temps_TWW = 5 THEN
870 Inbetriebnahme_Temps_TWW := 6;
871 END_IF;
872 IF TIRC302_clamp_ < Temp_max AND TIRC302_clamp_ > Temp_min AND Inbetriebnahme_Temps_TWW = 6 THEN
873 Inbetriebnahme_Temps_TWW := 7;
874 END_IF;
875 IF TIR301_clamp_ < Temp_max AND TIR301_clamp_ > Temp_min AND Inbetriebnahme_Temps_TWW = 7 THEN
876 Inbetriebnahme_Temps_TWW := 8;
877 END_IF;
878 IF TIRC305_clamp_ < Temp_max AND TIRC305_clamp_ > Temp_min AND Inbetriebnahme_Temps_TWW = 8 THEN
879 Inbetriebnahme_Temps_TWW := 9;
880 Inbetriebnahme_Szenario_TWW := 3;
881 FLAG_INBE_2_TWW := FALSE;
882 END_IF;
883 END_IF;
884 (*Ueberpruefen der STB-Prueftaste, wenn Temperaturen im geforderten Bereich*)
885 IF Inbetriebnahme_Szenario_TWW = 3 AND STB_pruef_TWW = TRUE THEN
886 Inbetriebnahme_Szenario_TWW := 4;
887 END_IF;
888 (*Ueberpruefen von EINER Verdrahtung - fuer reales System mit allen Anschluessen erweitern!!*)
889 IF Inbetriebnahme_Szenario_TWW = 4 AND TIRC401_clamp_ = 850.5 THEN (*in realer WUEST anstatt 220:
890 850.5 *)
891 Inbetriebnahme_Szenario_TWW := 5;
892 END_IF;
893 (*Ueberpruefen des Speichers*)
894 IF Inbetriebnahme_Szenario_TWW = 5 AND FLAG_INBE_5_TWW = FALSE THEN
895 SOC_soll := 80;
896 END_IF;
897 IF Inbetriebnahme_Szenario_TWW = 5 AND SOC > 78 THEN
898 Inbetriebnahme_Szenario_TWW := 6;
899 FLAG_INBE_5_TWW := TRUE;
900 END_IF;
901 IF Inbetriebnahme_Szenario_TWW = 6 THEN
902 SOC_soll := 20;
903 END_IF;
904 IF Inbetriebnahme_Szenario_TWW = 6 AND SOC < 22 THEN
905 (*Inbetriebnahme abgeschlossen*)
906 EXTERN_Inbetriebnahme_abgeschlossen_TWW := TRUE;
907 Ergebnis_funktionspruefung_TWW := 0;
908 FLAG_INBE_5_TWW := FALSE;
909 EXTERN_Inbetriebnahme_start_TWW := FALSE;
910 END_IF;
911 (*Stoppen der Funktionspruefung bei Abbruch von Inbetriebnahme:*)
912 ELSE start_funktionspruefung_1_TWW := FALSE;
913 END_IF;
914
915 (*Ueberpruefen, ob Fuellstands- oder Tempregelung*)
916 IF FLAG_2_TWW = TRUE THEN
917 MV301_soll := TIRC324_clamp_soll;
918 MV301_ist := TIRC324_clamp_;
```

```
919 ELSIF FLAG_2_TWW = FALSE THEN
920 MV301_soll := SOC_soll;
921 MV301_ist := SOC;
922 END_IF;
923
924
925 (*MV302_clamp := MV302_clamp_;*)
926 MV301_clamp := REAL_TO_WORD(MV301_clamp_);
927 P401_clamp := REAL_TO_WORD(P401_clamp_);
928 generator_clamp := REAL_TO_WORD(generator_clamp_);
929 END_IF;
930 END_PROGRAM
931
932
933 (* @NESTEDCOMMENTS := 'Yes' *)
934 (* @PATH := '' *)
935 (* @OBJECTFLAGS := '0, 8' *)
936 (* @SYMFILEFLAGS := '2048' *)
937 PROGRAM PLC_PRG_WUEST
938 VAR
939         FLAG_SET_OFFSETS_WUE: BOOL := FALSE;
940         TIRC102_clamp_: REAL;
941         TIRC103_clamp_: REAL;
942
943 (*Normalbetrieb*)
944         TC101_Kunde_soll: REAL := 100;
945         TC101_vertrag_soll : REAL;
946         TC101_Leitwarte_soll: REAL := 90;
947         TC101_clamp_soll: REAL;
948 (*RL-Begrenzung*)
949         TIRC103_clamp_max: REAL := 55;
950         TIRC103_clamp_zulaessig: REAL := 40;
951         dT_RLbegrenzung: REAL;
952         dT_RLbegrenzung_grenz: REAL := 10;
953         Temp_minderung_RLbegrenzung: INT := 5;
954         FLAG_dT: BOOL;
955 (*Leistungsbegrenzung*)
956         Timer_WMZ_WUE: TON;
957         Timer_step_sollwert_8: TON;
958         (*f_ueberschreitung_zulaessig_WUE: REAL := 1.1;*)
959         P_WMZ001_ARRAY20min: ARRAY[0..39] OF REAL;
960         Timer_WMZ_IN_WUE: BOOL := TRUE;
961         Timer_WMZ_Q_WUE: BOOL;
962         Timer_step_IN: BOOL := TRUE;
963         Timer_step_Q: BOOL;
964         P_WMZ001_20min: REAL;
965         Zaehler_leistung_8: INT;
966         P_ARRAY_WUE: INT;
967         P_WMZ001_20min_mittel: REAL;
968         FLAG_8_WUE: BOOL;
969         Zaehler_8: INT;
970         Zaehler_8_alt: INT;
971         MV101_soll_alt: REAL;
972         delta_Zaehler_8: INT;
973 (*RL-Regelung*)
974         EXTERN_RLTempRegelung: BOOL;
975         EXTERN_RLTemp_soll: REAL := 40;
976         FLAG_6_WUE: BOOL;
977 (*Funktionspruefung*)
978         (*EXTERN_DO_funktionspruefung_WUE: BOOL;*)
```

```
979             start_funktionspruefung_5_WUE: BOOL;
980             FLAG_5_WUE: BOOL;
981             Massenstrom_ges_WUE: REAL;
982             Massenstrom_max_WUE: REAL := 0.7;
983             Massenstrom_min_WUE: REAL := 0;
984             Zaehler_5_WUE: INT := 0;
985             Ergebnis_funktionspruefung_WUE: WORD;
986 (*Kommunikationsausfall*)
987             (*Timer_Toggle_WUE: TON;
988             IN_Timer_Toggle_WUE: BOOL;*)
989 (*Inbetriebnahme*)
990             EXTERN_Inbetriebnahme_start_WUE: BOOL;
991             EXTERN_Inbetriebnahme_abgeschlossen_WUE: BOOL;
992             Inbetriebnahme_Szenario_WUE: INT;
993             start_funktionspruefung_1_WUE: BOOL;
994             Inbetriebnahme_Temps_WUE: INT;
995             STB_pruef_WUE: BOOL;
996             FLAG_INBE_2_WUE: BOOL;
997             FLAG_INBE_5_WUE: BOOL;
998             Temp_min: REAL;
999             Temp_max: REAL;
1000 (*Netzkollaps*)
1001            FLAG_4_WUE: BOOL;
1002            TC101_Netzkollaps_soll: REAL := 60;
1003            TC101_Netzkollaps_berechnet_soll: REAL;
1004            T_WUT_graedigkeit: REAL := 5;
1005
1006 END_VAR
1007 (* @END_DECLARATION := '0' *)
1008 (*Einlesen der Temperaturen an den Klemmen und in Grad Celsius umrechnen*)
1009 TC101_clamp_ := WORD_TO_REAL(TC101_clamp) / Skalierung_Temp_clamp;
1010 TIRC102_clamp_ := WORD_TO_REAL(TIRC102_clamp) / Skalierung_Temp_clamp;
1011 TIRC103_clamp_ := WORD_TO_REAL(TIRC103_clamp) / Skalierung_Temp_clamp;
1012
1013 (*Skalierung der Konstanten - Kelvin/Grad Celsius, wird nur beim ersten Programmaufruf durchgeführt*)
1014 IF FLAG_SET_OFFSETS_WUE = FALSE THEN
1015 TC101_Kunde_soll := TC101_Kunde_soll + OFFSET_Kelvin;
1016 TC101_Leitwarte_soll := TC101_Leitwarte_soll + OFFSET_Kelvin;
1017 TIRC103_clamp_max := TIRC103_clamp_max + OFFSET_Kelvin;
1018 TIRC103_clamp_zulaessig := TIRC103_clamp_zulaessig + OFFSET_Kelvin;
1019 EXTERN_RLTemp_soll := EXTERN_RLTemp_soll + OFFSET_Kelvin;
1020 TIRC_max := TIRC_max + OFFSET_Kelvin;
1021 TIRC_min := TIRC_min + OFFSET_Kelvin;
1022 TC101_Netzkollaps_soll := TC101_Netzkollaps_soll + OFFSET_Kelvin;
1023 END_IF;
1024 FLAG_SET_OFFSETS_WUE := TRUE;
1025
1026
1027 (*----------Normalbetrieb----------*)
1028 TC101_vertrag_soll := 70; (* In Vertrag bestimmt*)
1029 (*Temperaturprioritaeten*)
1030 IF TC101_Kunde_soll < TC101_vertrag_soll THEN
1031 TC101_clamp_soll := TC101_Kunde_soll;
1032 ELSIF T_Netz_ist < TC101_vertrag_soll THEN
1033 TC101_clamp_soll := T_Netz_ist - 5;
1034 ELSIF TC101_Leitwarte_soll > TC101_vertrag_soll THEN
1035 TC101_clamp_soll := TC101_vertrag_soll;
1036 ELSE TC101_clamp_soll := TC101_Leitwarte_soll;
1037 END_IF;
1038
```

```
1039 Szenario_WUE := 10;
1040 MV101_soll := TC101_clamp_soll;
1041 MV101_ist := TC101_clamp_;
1042 (*Regelparameter fuer Regler pid_MV101, falls nicht anders in Szenario definiert*)
1043 IF FLAG_JARVIS = TRUE THEN
1044 Kp_MV101 := 0.00005;
1045 Tn_MV101 := 1000000;
1046 Tv_MV101 := 0;
1047 END_IF;
1048 IF FLAG_JARVIS = FALSE THEN
1049 Kp_MV101 := 300;
1050 Tn_MV101 := 40;
1051 Tv_MV101 := 0;
1052 END_IF;
1053 (*----------RL-Begrenzung----------*)
1054 (*Berechnung delta_T sekundaer WUT Heizkreise*)
1055 dT_RLbegrenzung := TC101_clamp_ - TIRC102_clamp_;
1056 IF dT_RLbegrenzung > dT_RLbegrenzung_grenz THEN
1057 FLAG_dT := TRUE;
1058 ELSE FLAG_dT := FALSE;
1059 END_IF;
1060 (*Zaehler_gesamt wird benoetigt, um System einzupendeln.
1061  Nach 300 Zeitschritten wird Abfrage der Szenarien gestartet*)
1062 Zaehler_gesamt := Zaehler_gesamt + 1;
1063 IF Zaehler_gesamt > 61 THEN
1064 IF TIRC103_clamp_ > TIRC103_clamp_max OR FLAG_9_WUE = TRUE THEN
1065 Szenario_WUE := 9;
1066 FLAG_9_WUE := TRUE;
1067 END_IF;
1068 IF TIRC103_clamp_ < TIRC103_clamp_zulaessig AND FLAG_dT = TRUE THEN
1069 FLAG_9_WUE := FALSE;
1070 Zaehler_WUE_9 := 0;
1071 END_IF;
1072 (*Solltemperatur TC101_soll wird nach jeweils 300 Zeitschritten verringert,
1073  um RL-Temp zu verringern*)
1074 IF FLAG_9_WUE = TRUE THEN
1075 MV101_soll := TC101_clamp_soll - Temp_minderung_RLbegrenzung;
1076 IF Zaehler_WUE_9 = 300 THEN
1077 Temp_minderung_RLbegrenzung := Temp_minderung_RLbegrenzung + 3;
1078 END_IF;
1079 Zaehler_WUE_9 := Zaehler_WUE_9 + 1;
1080 IF Zaehler_WUE_9 = 301 THEN
1081 Zaehler_WUE_9 := 0;
1082 END_IF;
1083 ELSIF FLAG_9_WUE = FALSE THEN
1084 Temp_minderung_RLbegrenzung := 5;
1085 END_IF;
1086
1087 (*Temp_minderung_RLbegrenzung soll nicht geaendert werden, falls szenario 8 - 1 zwischenzeitlich
1088 aktiv ist*)
1089 IF Szenario_WUE_alt < 9 THEN
1090 Temp_minderung_RLbegrenzung := 5;
1091 END_IF;
1092
1093 (*----------Leistungsbegrenzung----------*)
1094 (*Annahme: Oeffnung Mischventil entspricht Durchfluss in l/s, in realem System kommt P von WMZ*)
1095 (*Berechnung der Leistung P_WMZ001 in PLC_PRG_TWW*)
1096
1097 (*Timer fuer die Ableserate von WMZ, alle 30s wird Leistung ausgelesen*)
1098 Timer_WMZ_WUE(IN:=Timer_WMZ_IN_WUE, PT := T#30s);
```

```
1099 Timer_WMZ_Q_WUE := Timer_WMZ_WUE.Q;
1100 Timer_WMZ_IN_WUE := TRUE;
1101 IF Timer_WMZ_Q_WUE = TRUE THEN
1102 Timer_WMZ_IN_WUE := FALSE;
1103 (*Speichern der Leistung in Array*)
1104 P_WMZ001_ARRAY20min[P_ARRAY_WUE] := P_WMZ001;
1105 P_ARRAY_WUE := P_ARRAY_WUE + 1;
1106 Zaehler_leistung_8 := 0;
1107 P_WMZ001_20min := 0;
1108 (*Berechnung Mittelwert Leistung 20min*)
1109 WHILE Zaehler_leistung_8 < 40 DO
1110 P_WMZ001_20min := P_WMZ001_20min + P_WMZ001_ARRAY20min[Zaehler_leistung_8];
1111 Zaehler_leistung_8 := Zaehler_leistung_8 + 1;
1112 END_WHILE;
1113 END_IF;
1114 IF P_ARRAY_WUE = 40 THEN
1115 P_ARRAY_WUE := 0;
1116 END_IF;
1117 P_WMZ001_20min_mittel := P_WMZ001_20min/40;
1118 Grenzwert_kulanz := P_WMZ001_zulaessig * f_ueberschreitung_zulaessig;
1119 (*Abfrage Leistungsbegrenzung*)
1120 IF P_WMZ001_20min_mittel > Grenzwert_kulanz OR FLAG_8_WUE = TRUE THEN
1121 FLAG_8_WUE := TRUE;
1122 END_IF;
1123 IF P_WMZ001_20min_mittel < P_WMZ001_zulaessig THEN
1124 FLAG_8_WUE := FALSE;
1125 END_IF;
1126
1127 (*Temperatursollwert sinkt um 1grad pro 30s, damit Leistungsabnahme sinkt*)
1128 IF FLAG_8_WUE = TRUE THEN
1129 Timer_step_sollwert_8(IN:=Timer_step_IN, PT := T#30s);
1130 Timer_step_Q := Timer_step_sollwert_8.Q;
1131 Timer_step_IN := TRUE;
1132 IF Timer_step_Q = TRUE AND FLAG_2_WUE = FALSE THEN
1133 Zaehler_8 := Zaehler_8 + 1;
1134 Timer_step_IN := FALSE;
1135 END_IF;
1136 MV101_ist := TC101_clamp_;
1137 MV101_soll := TC101_clamp_soll - Zaehler_8;
1138 Szenario_WUE := 8;
1139 Zaehler_8_alt := Zaehler_8;
1140 MV101_soll_alt := MV101_soll;
1141 END_IF;
1142 IF FLAG_8_WUE = FALSE THEN
1143 Zaehler_8 := 0;
1144 END_IF;
1145 delta_Zaehler_8 := Zaehler_8 - Zaehler_8_alt;
1146
1147 (*----------Ruecklauftemperaturregelung----------*)
1148 IF EXTERN_RLTempRegelung = TRUE THEN
1149 MV101_soll := EXTERN_RLTemp_soll;
1150 MV101_ist := TIRC103_clamp_;
1151 IF FLAG_JARVIS=TRUE THEN
1152 Kp_MV101 := 0.000002; (*0.000001, alternative Parametereinstellung*)
1153 Tn_MV101 := 500; (*100*)
1154 Tv_MV101 := 100; (*100*)
1155 END_IF;
1156 Szenario_WUE := 6;
1157 FLAG_6_WUE := TRUE;
1158 ELSIF EXTERN_RLTempRegelung = FALSE THEN
```

```
1159 FLAG_6_WUE := FALSE;
1160 END_IF;
1161 (*----------Funktionspruefung----------*)
1162 IF EXTERN_DO_funktionspruefung_WUE = TRUE THEN
1163 (*Externe Aktivierung von Funktionspruefung fuer iWUEST ohne TWW nicht moeglich wegen WMZ fuer
1164 komplette Anlage*)
1165 start_funktionspruefung_5_WUE := TRUE;
1166 Szenario_WUE := 5;
1167 FLAG_5_WUE := TRUE;
1168 ELSIF EXTERN_DO_funktionspruefung_WUE = FALSE AND EXTERN_Inbetriebnahme_start_WUE = FALSE THEN
1169 start_funktionspruefung_5_WUE := FALSE;
1170 FLAG_5_WUE := FALSE;
1171 Ergebnis_funktionspruefung_WUE := 0;
1172 STATUS_funktionspruefung_WUE := 0;
1173 END_IF;
1174 IF start_funktionspruefung_5_WUE = TRUE OR start_funktionspruefung_1_WUE = TRUE THEN
1175 (*In realem System wird Massenstrom durch WMZ geliefert*)
1176 Massenstrom_ges_WUE :=  ((MV101_clamp_ - 0.004)/0.016)*0.7;
1177 IF FLAG_JARVIS = FALSE THEN
1178 Massenstrom_ges_WUE :=  (MV101_clamp/32767)*0.7;
1179 END_IF;
1180 (*Ventile schliesssen(via SETOUTPUT):*)
1181 IF STATUS_funktionspruefung_WUE <> 2 AND STATUS_funktionspruefung_WUE <> 3 THEN
1182 STATUS_funktionspruefung_WUE := 1;
1183 END_IF;
1184 IF STATUS_funktionspruefung_WUE = 1 THEN
1185 Zaehler_5_WUE := Zaehler_5_WUE + 1;
1186 TC101_clamp_ := TIRC_max;
1187 IF Massenstrom_ges_WUE = Massenstrom_min_WUE AND Zaehler_5_WUE > 30 THEN
1188 STATUS_funktionspruefung_WUE := 2;
1189 Zaehler_5_WUE := 0;
1190 END_IF;
1191 END_IF;
1192 (*ventil oeffnen(via SETOUTPUT) : *)
1193 IF STATUS_funktionspruefung_WUE = 2  THEN
1194 Zaehler_5_WUE := Zaehler_5_WUE + 1;
1195 TC101_clamp_ := TIRC_min;
1196 IF Massenstrom_ges_WUE = Massenstrom_max_WUE AND Zaehler_5_WUE > 30 THEN
1197 STATUS_funktionspruefung_WUE := 3;
1198 END_IF;
1199 END_IF;
1200 IF STATUS_funktionspruefung_WUE = 3 THEN
1201 Ergebnis_funktionspruefung_WUE := 200;
1202 (*Naechster Test der  Inbetriebnahme wird aktiviert*)
1203 IF Inbetriebnahme_Szenario_WUE = 1 THEN
1204 Inbetriebnahme_Szenario_WUE := 2;
1205 END_IF;
1206 STATUS_funktionspruefung_WUE := 0;
1207 Zaehler_5_WUE := 0;
1208 (*Schliesssen der Ventile gescheitert*)
1209 ELSIF Zaehler_5_WUE = 500 AND STATUS_funktionspruefung_WUE = 1 THEN
1210 Ergebnis_funktionspruefung_WUE := 40001;
1211 STATUS_funktionspruefung_WUE := 1;
1212 Zaehler_5_WUE := 0;
1213 (*Oeffnen der Ventile gescheitert*)
1214 ELSIF Zaehler_5_WUE = 500 AND STATUS_funktionspruefung_WUE = 2 THEN
1215 Ergebnis_funktionspruefung_WUE := 40002;
1216 STATUS_funktionspruefung_WUE := 1;
1217 Zaehler_5_WUE := 0;
1218 END_IF;
```

```
1219 END_IF;
1220
1221 (*----------------------------------------Netzkollaps----------------------------------------*)
1222 IF EXTERN_Netzkollaps = TRUE THEN
1223 FLAG_4_WUE := TRUE;
1224 Szenario_WUE := 4;
1225 MV101_ist := TC101_clamp_;
1226 TC101_Netzkollaps_berechnet_soll := T_Netz_mittel - T_WUT_graedigkeit;
1227 (*Sollwert fuer Netzkollaps kann hinterlegt sein, ansonsten wie oben berechnet*)
1228 IF TC101_Netzkollaps_soll <> 0 THEN
1229 MV101_soll := TC101_Netzkollaps_soll;
1230 ELSE MV101_soll := TC101_Netzkollaps_berechnet_soll;
1231 END_IF;
1232 (*Temperaturprioritaeten*)
1233
1234 IF TC101_Kunde_soll < TC101_vertrag_soll THEN
1235 TC101_clamp_soll := TC101_Kunde_soll;
1236 ELSIF MV101_soll < TC101_vertrag_soll THEN
1237 TC101_clamp_soll := MV101_soll;
1238 ELSIF TC101_Leitwarte_soll > TC101_vertrag_soll THEN
1239 TC101_clamp_soll := TC101_vertrag_soll;
1240 ELSE TC101_clamp_soll := TC101_Leitwarte_soll;
1241 END_IF;
1242 MV101_soll := TC101_clamp_soll;
1243 ELSE FLAG_4_WUE := FALSE;
1244 END_IF;
1245
1246 (*----------------------------------Kommunikationsausfall----------------------------------*)
1247 (* Kommunikationsausfall wird in PLC_PRG_TWW ueberprueft und von FLAG_3 angezeigt*)
1248
1249 IF FLAG_3 = TRUE THEN
1250 Szenario_WUE := 3;
1251 MV101_ist := TC101_clamp_;
1252 MV101_soll := T_Netz_ist - T_WUT_graedigkeit;
1253
1254 (*Temperaturprioritaeten*)
1255 IF TC101_Kunde_soll < TC101_vertrag_soll THEN
1256 TC101_clamp_soll := TC101_Kunde_soll;
1257 ELSIF MV101_soll < TC101_vertrag_soll THEN
1258 TC101_clamp_soll := MV101_soll;
1259 ELSIF TC101_Leitwarte_soll > TC101_vertrag_soll THEN
1260 TC101_clamp_soll := TC101_vertrag_soll;
1261 ELSE TC101_clamp_soll := TC101_Leitwarte_soll;
1262 END_IF;
1263 MV101_soll := TC101_clamp_soll;
1264 END_IF;
1265
1266 (*Szenario2: Warmhaltung in SETOUTPUT(PRG*)
1267
1268 (*-------------------------------------Inbetriebnahme-------------------------------------*)
1269 (*Externe Aktivierung von Inbetriebnahme fuer iWUEST ohne TWW nicht moeglich wegen WMZ fuer komplette
1270 Anlage*)
1271 IF EXTERN_Inbetriebnahme_start_WUE = TRUE AND EXTERN_Inbetriebnahme_abgeschlossen_WUE = FALSE THEN
1272 (*Pingtest durch Empfangen des Bit EXTERN_Inbetriebnahme_start_WUE bestanden*)
1273 Szenario_WUE := 1;
1274 IF Ergebnis_funktionspruefung_TWW <> 200 THEN
1275 Inbetriebnahme_Szenario_WUE := 1;
1276 (*start_funktionspruefung_1_WUE := TRUE;*)
1277 start_funktionspruefung_1_TWW := TRUE;
1278 FLAG_INBE_2_WUE := TRUE;
```

```
1279 END_IF;
1280 IF Ergebnis_funktionspruefung_TWW = 200 AND FLAG_INBE_2_WUE = TRUE THEN
1281 Inbetriebnahme_Szenario_WUE := 2;
1282 (*start_funktionspruefung_1_WUE := FALSE;*)
1283 start_funktionspruefung_1_TWW := FALSE;
1284 Temp_min := 0 + OFFSET_Kelvin;
1285 Temp_max := 100 + OFFSET_Kelvin;
1286 IF TC101_clamp_ < Temp_max AND TC101_clamp_ > Temp_min THEN
1287 Inbetriebnahme_Temps_WUE := 1;
1288 END_IF;
1289 IF TIRC102_clamp_ < Temp_max AND TIRC102_clamp_ > Temp_min AND Inbetriebnahme_Temps_WUE = 1 THEN
1290 Inbetriebnahme_Temps_WUE := 2;
1291 END_IF;
1292 IF TIRC103_clamp_ < Temp_max AND TIRC103_clamp_ > Temp_min AND Inbetriebnahme_Temps_WUE = 2 THEN
1293 Inbetriebnahme_Szenario_WUE := 3;
1294 FLAG_INBE_2_WUE := FALSE;
1295 END_IF;
1296 END_IF;
1297 IF Inbetriebnahme_Szenario_WUE = 3 AND STB_pruef_WUE = TRUE THEN
1298 Inbetriebnahme_Szenario_WUE := 4;
1299 END_IF;
1300 IF Inbetriebnahme_Szenario_WUE = 4 AND TIRC102_clamp_ = 216 THEN (*in realer WUEST anstatt 216:
1301 850.5 *)
1302 EXTERN_Inbetriebnahme_abgeschlossen_WUE := TRUE;
1303 Ergebnis_funktionspruefung_TWW := 0;
1304 FLAG_INBE_5_WUE := FALSE;
1305 EXTERN_Inbetriebnahme_start_WUE := FALSE;
1306 END_IF;
1307 (*Stoppen der Funktionspruefung bei Abbruch von Inbetriebnahme:*)
1308 ELSE start_funktionspruefung_1_WUE := FALSE;
1309 END_IF;
1310 (*----------Ueberpruefung Temperatur- und Druckwaechter----------*)
1311 IF TS_clamp = TRUE OR TZ_clamp = TRUE THEN
1312 Szenario_WUE := 0;
1313 FLAG_0_WUE := TRUE;
1314 ELSIF TS_clamp = FALSE AND TZ_clamp = FALSE THEN
1315 FLAG_0_WUE := FALSE;
1316 END_IF;
1317
1318 MV101_clamp := REAL_TO_WORD(MV101_clamp_);
1319 END_IF;
1320 END_PROGRAM
1321
1322
1323 (* @NESTEDCOMMENTS := 'Yes' *)
1324 (* @PATH := '' *)
1325 (* @OBJECTFLAGS := '0, 8' *)
1326 (* @SYMFILEFLAGS := '2048' *)
1327 PROGRAM SETOUTPUT
1328 VAR
1329 (*Warmhaltung*)
1330         MV101_offen_min:REAL := 0.03;
1331         Oeffnung_Ventil: REAL;
1332         MV101_offen_zulaessig: REAL := 0.05;
1333         Ausgangssignal_max_jarvis: REAL := 0.02;
1334         Ausgangssignal_min_jarvis: REAL := 0.004;
1335         Ausgangssignal_max_codesys: REAL := 32767;
1336         Ausgangssignal_min_codesys: REAL := 0;
1337         Signal_min_oeffnung: REAL;
1338
```

```
1339          Ausgangssignal_max:REAL;
1340          Ausgangssignal_min:REAL;
1341  END_VAR
1342  (* @END_DECLARATION := '0' *)
1343  (*Einlesen der Temperaturen an den Klemmen und in Grad Celsius umrechnen*)
1344  TIRC302_clamp_ := WORD_TO_REAL(TIRC302_clamp) / Skalierung_Temp_clamp;
1345  TIRC322_clamp_ := WORD_TO_REAL(storage1_TIRC322) / Skalierung_Temp_clamp;
1346
1347  (*----------------------------------Warmhaltung WUE----------------------------------*)
1348  (*Annahme: minimale Oeffnung ist 3Prozent, Ausstieg aus Szenario bei 5Prozent*)
1349  IF FLAG_JARVIS = FALSE THEN
1350  Ausgangssignal_max := Ausgangssignal_max_codesys;
1351  Ausgangssignal_min := Ausgangssignal_min_codesys;
1352  END_IF;
1353  IF FLAG_JARVIS = TRUE THEN
1354  Ausgangssignal_max := Ausgangssignal_max_jarvis;
1355  Ausgangssignal_min := Ausgangssignal_min_jarvis;
1356  END_IF;
1357  (*Oeffnungswinkel in %*)
1358  Signal_min_oeffnung := Ausgangssignal_min + (MV101_offen_zulaessig*(Ausgangssignal_max-
1359                         Ausgangssignal_min));
1360  (*Oeffnung_Ventil := ((MV101_clamp-Ausgangssignal_min) / (Ausgangssignal_max - Ausgangssignal_min));*)
1361  Oeffnung_Ventil := ((MV101_clamp_-Ausgangssignal_min) / (Ausgangssignal_max - Ausgangssignal_min));
1362  IF Zaehler_gesamt > 61 THEN
1363  IF Szenario_WUE > 1 AND Szenario_WUE <> 5 THEN
1364  IF Oeffnung_Ventil < MV101_offen_min OR FLAG_2_WUE = TRUE THEN
1365  FLAG_2_WUE := TRUE;
1366  END_IF;
1367  IF Oeffnung_Ventil > MV101_offen_zulaessig THEN
1368  FLAG_2_WUE := FALSE;
1369  END_IF;
1370  IF FLAG_2_WUE = TRUE THEN
1371  Szenario_WUE := 2;
1372  MV101_clamp := REAL_TO_WORD(Signal_min_oeffnung);
1373  (*untere Zeile war in Code von Simulation, musste raus fuer HW-TU. kann wegbleiben??*)
1374  (*MV101_clamp_ := MV101_clamp;*)
1375  END_IF;
1376  END_IF;
1377
1378  (*-------------------------Minimale Ventiloeffnung Szenario 7 TWW-------------------------*)
1379  IF Szenario_TWW = 7 THEN
1380  MV301_clamp := REAL_TO_WORD(Ausgangssignal_min + (Ausgangssignal_max/20));
1381  MV301_clamp_ := MV301_clamp;
1382  END_IF;
1383
1384
1385  (*Uebergabe an Regler*)
1386
1387  IF FLAG_JARVIS = TRUE THEN
1388  IF TIRC302_clamp_ > TIRC322_clamp_ THEN
1389  MV302_clamp := REAL_TO_WORD(P401_clamp-0.000001);
1390  ELSE MV302_clamp := REAL_TO_WORD( 0.004001);
1391  END_IF;
1392  END_IF;
1393  IF FLAG_JARVIS = FALSE THEN
1394  IF TIRC302_clamp_ > TIRC322_clamp_ THEN
1395  MV302_clamp := REAL_TO_WORD(Ausgangssignal_max);
1396  ELSE MV302_clamp := REAL_TO_WORD(Ausgangssignal_min);
1397  END_IF;
1398  END_IF;
```

```
1399
1400 Szenario_WUE_alt := Szenario_WUE;
1401
1402 (*zu Funktionspruefung und Inbetriebnahme TWW*)
1403 IF STATUS_funktionspruefung_TWW = 1  THEN
1404 MV101_clamp := REAL_TO_WORD(Ausgangssignal_min);
1405 MV302_clamp := REAL_TO_WORD(Ausgangssignal_min);
1406 MV301_clamp := REAL_TO_WORD(Ausgangssignal_min);
1407 MV101_clamp_ := MV101_clamp;
1408 MV302_clamp_ := MV302_clamp;
1409 MV301_clamp_ := MV301_clamp;
1410 ELSIF STATUS_funktionspruefung_TWW = 2  THEN
1411 MV101_clamp := REAL_TO_WORD(Ausgangssignal_max);
1412 MV302_clamp := REAL_TO_WORD(Ausgangssignal_max);
1413 MV301_clamp := REAL_TO_WORD(Ausgangssignal_max);
1414 MV101_clamp_ := MV101_clamp;
1415 MV302_clamp_ := MV302_clamp;
1416 MV301_clamp_ := MV301_clamp;
1417 END_IF;
1418
1419 IF FLAG_0_WUE = TRUE THEN
1420 MV101_clamp := REAL_TO_WORD(Ausgangssignal_min);
1421 MV302_clamp := REAL_TO_WORD(Ausgangssignal_min);
1422 MV301_clamp := REAL_TO_WORD(Ausgangssignal_min);
1423 MV101_clamp_ := MV101_clamp;
1424 MV302_clamp_ := MV302_clamp;
1425 MV301_clamp_ := MV301_clamp;
1426 END_IF;
1427
1428 END_IF;
1429
1430 (*----------------------AUSGABE SOC und SZENARIEN,
1431  EINKOMMENTIEREN fuer Ausgabe der Werte im Terminal----------------*)
1432 (*IF Szenario_TWW = 2 THEN
1433 Zaehler_gesamt := Zaehler_gesamt + 1;
1434 END_IF;*)
1435 (*SOC_data[Zaehler_gesamt] := SOC;
1436 SOC_soll_data[Zaehler_gesamt] := SOC_soll;
1437 Szenario_WUE_data[Zaehler_gesamt] := Szenario_WUE;
1438 Szenario_TWW_data[Zaehler_gesamt] := Szenario_TWW;
1439 P_WMZ001_20min_mittel_data[Zaehler_gesamt] := P_WMZ001_20min_mittel;
1440 print("Zaehler_gesamt=", Zaehler_gesamt)
1441 IF Zaehler_gesamt = 5001 THEN (*2815*)
1442 print("SOC=", SOC_data)
1443 print("SOC_soll=", SOC_soll_data)
1444 print("Szenarien_TWW=", Szenario_TWW_data)
1445 print("Szenarien_WUE=", Szenario_WUE_data)
1446 END_IF;*)
1447 END_PROGRAM
1448
1449 (* @NESTEDCOMMENTS := 'Yes' *)
1450 (* @GLOBAL_VARIABLE_LIST := 'Globale_Variablen' *)
1451 (* @PATH := '' *)
1452 (* @OBJECTFLAGS := '0, 8' *)
1453 (* @SYMFILEFLAGS := '2048' *)
1454 VAR_GLOBAL
1455         Skalierung_Temp_clamp: INT;
1456         OFFSET_Kelvin: REAL;
1457         TIRC401_clamp_soll : REAL := 60;
1458         MV101_ist: REAL;
```

```
1459          MV101_soll: REAL;
1460          Kp_MV101: REAL;
1461          Tn_MV101: REAL;
1462          Tv_MV101: REAL;
1463          MV301_soll: REAL;
1464          MV301_ist: REAL;
1465          TIRC401_clamp_: REAL;
1466          TIRC302_clamp_: REAL;
1467          TIRC322_clamp_: REAL;
1468          SOC: REAL;
1469          SOC_soll: REAL;
1470          c_p: REAL := 4190;
1471          (*pid_MV301: PID;
1472          pid_MV101: PID;
1473          pid_P401: PID;*)
1474          MV301_clamp_: REAL;
1475          MV302_clamp_: WORD;
1476          TC101_clamp_: REAL;
1477          P401_clamp_: REAL;
1478          MV101_clamp_: REAL;
1479          pid_generator: PID;
1480          generator_clamp_: REAL;
1481          generator_clamp: WORD;
1482          T_Netz_soll: REAL := 90;
1483          T_Netz_ist: REAL;
1484          T_Netz_mittel: REAL := 70;
1485          P_WMZ001: REAL;
1486          P_WMZ001_zulaessig: REAL := 105;
1487          f_ueberschreitung_zulaessig: REAL := 1.1;
1488          Grenzwert_kulanz: REAL;
1489          FLAG_3: BOOL;
1490          STATUS_funktionspruefung_TWW: INT := 0;
1491          STATUS_funktionspruefung_WUE: INT := 0;
1492          FLAG_2_WUE: BOOL;
1493          FLAG_2_TWW: BOOL;
1494          Szenario_WUE: INT;
1495          Szenario_TWW: INT;
1496          TIRC_max: REAL := 100;
1497          TIRC_min: REAL := 0;
1498          FLAG_Timer_Fuellstandsbegrenzung: BOOL;
1499          EXTERN_DO_funktionspruefung_WUE: BOOL := FALSE;
1500          EXTERN_Inbetriebnahme_start_WUE: BOOL;
1501          start_funktionspruefung_1_TWW: BOOL;
1502          Ergebnis_funktionspruefung_TWW: WORD;
1503          EXTERN_Netzkollaps: BOOL;
1504          EXTERN_Toggle: BOOL;
1505          Zaehler_WUE_9: INT;
1506          FLAG_9_WUE: BOOL;
1507          Szenario_WUE_alt: INT;
1508  (*Waechter*)
1509          FLAG_0_WUE: BOOL;
1510          Zaehler_gesamt: INT := 0;
1511  (*Fuer Jarvis aendern!!!---------->*)
1512          FLAG_JARVIS: BOOL := FALSE; (*=TRUE in Jarvis*)
1513          (*Toggle_data : ARRAY[0..5000] OF REAL;
1514          Zaehler_gesamt: INT;
1515          SOC_data : ARRAY[0..5000] OF REAL;
1516          SOC_soll_data : ARRAY[0..5000] OF REAL;
1517          Szenario_WUE_data: ARRAY[0..5000] OF REAL;
1518          Szenario_TWW_data: ARRAY[0..5000] OF REAL;
```

```
1519          P_WMZ001_20min_mittel_data: ARRAY[0..5000] OF REAL;
1520          P_WMZ001_data: ARRAY[0..5000] OF REAL;
1521          P_WMZ002_data: ARRAY[0..5000] OF REAL;
1522          TS_clamp: BOOL := FALSE;
1523          TZ_clamp: BOOL := FALSE; beide einkommentieren*)
1524 (*----------------------------------------------------------------*)
1525 END_VAR
1526 (* @OBJECT_END := 'Globale_Variablen' *)
1527 (* @CONNECTIONS := Globale_Variablen
1528 FILENAME : ''
1529 FILETIME : 0
1530 EXPORT : 0
1531 NUMOFCONNECTIONS : 0
1532 *)
1533
1534 (* @NESTEDCOMMENTS := 'Yes' *)
1535 (* @GLOBAL_VARIABLE_LIST := 'Variablen_Konfiguration' *)
1536 (* @PATH := '' *)
1537 (* @OBJECTFLAGS := '0, 8' *)
1538 (* @SYMFILEFLAGS := '2048' *)
1539 VAR_CONFIG
1540 END_VAR
1541
1542 (* @OBJECT_END := 'Variablen_Konfiguration' *)
1543 (* @CONNECTIONS := Variablen_Konfiguration
1544 FILENAME : ''
1545 FILETIME : 0
1546 EXPORT : 0
1547 NUMOFCONNECTIONS : 0
1548 *)
1549
1550 (* @NESTEDCOMMENTS := 'Yes' *)
1551 (* @GLOBAL_VARIABLE_LIST := 'Variablen_Konfiguration_1' *)
1552 (* @PATH := '' *)
1553 (* @OBJECTFLAGS := '0, 8' *)
1554 (* @SYMFILEFLAGS := '2048' *)
1555 VAR_CONFIG
1556 END_VAR
1557
1558 (* @OBJECT_END := 'Variablen_Konfiguration_1' *)
1559 (* @CONNECTIONS := Variablen_Konfiguration_1
1560 FILENAME : ''
1561 FILETIME : 0
1562 EXPORT : 0
1563 NUMOFCONNECTIONS : 0
1564 *)
1565
1566
1567 _ALARMCONFIG
1568 _ALARMCONFIGNEXTTEXTID : 10002
1569 _ALARMCONFIGFORMATS : 'HH$':$'mm$':$'ss','dd$'-$'MM$'-$'yyyy'
1570 _ALARMCLASSLIST : 1
1571 _ALARMCLASSID : 0
1572 _ALARMCLASSACKTYPE : 0
1573 _ALARMCLASSNAME : 'DEFAULT'
1574 _ALARMCLASSDESCRIPTION : ''
1575 _ALARMCLASSBGCOLORS : 16777215,16777215,16777215
1576 _ALARMCLASSTEXTCOLORS : 3394560,255,16711680
1577 _ALARMCLASSBITMAPS : '','',''
1578 _ALARMACTIONLIST : 0
```

```
(* @ALARMCLASSRESETCOLORS := '_ALARMCLASSRESETCOLORS: 33023,16777215' *)
(* @ALARMCLASSRESETBITMAP := '_ALARMCLASSRESETBITMAP: $'$'' *)
_ALARMGROUPLISTNAME : 'System'
_ALARMGROUPPATH : 'System'
_ALARMGROUPLIST : 0
_VISUALSETTINGSFLAGS : 0,0,0,0
_VISUALSETTINGSFLAGS : '','',''
_VISUALSETTINGSDYNTEXTFILECOUNT : 0

(* @ALARMCONFIGFLAGS := '_ALARMCONFIGFLAGS: 4' *)
(* @ALARMCONFIGGLOBALDB_STR := '_ALARMCONFIGGLOBALDB_STRINGS: $'$',$'$',$'$',$'$',$'$'' *)
(* @ALARMCONFIGGLOBALDB_NUM := '_ALARMCONFIGGLOBALDB_NUMBERS: 0,0' *)
_END_ALARMCONFIG


_WORKSPACE
_GLOBALVISUALSETTINGS
_VISUALSETTINGSFLAGS : 0,0,0,0
_VISUALSETTINGSFLAGS : '','',''
_VISUALSETTINGSDYNTEXTFILECOUNT : 0
_VISUALBITMAPLISTCOUNT : 0
_END_GLOBALVISUALSETTINGS
_END_WORKSPACE


LIBRARY
C:\Program Files (x86)\WAGO Software\CoDeSys V2.3\Targets\WAGO\Libraries\IO_IPC\Util.lib 2.12.10
14:48:34
(* @LIBRARYSYMFILEINFO := '0' *)
NumOfPOUs: 21
BCD_TO_INT: 2048
BLINK: 2048
CHARCURVE: 2048
DERIVATIVE: 2048
EXTRACT: 2048
GEN: 2048
HYSTERESIS: 2048
INT_TO_BCD: 2048
INTEGRAL: 2048
LIMITALARM: 2048
PACK: 2048
PD: 2048
PID: 2048
PUTBIT: 2048
RAMP_INT: 2048
RAMP_REAL: 2048
STATISTICS_INT: 2048
STATISTICS_REAL: 2048
UNPACK: 2048
VARIANCE: 2048
Version_Util: 2048
NumOfGVLs: 1
Globale_Variablen: 0
END_LIBRARY

LIBRARY
C:\Program Files (x86)\WAGO Software\CoDeSys V2.3\Targets\WAGO\Libraries\PFC200\SysLibSockets.lib
22.5.14 16:16:32
(* @LIBRARYSYMFILEINFO := '0' *)
NumOfPOUs: 26
```

```
1639 SysSockAccept: 0
1640 SysSockBind: 0
1641 SysSockClose: 0
1642 SysSockConnect: 0
1643 SysSockCreate: 0
1644 SysSockGetHostByName: 0
1645 SysSockGetHostName: 0
1646 SysSockGetLastError: 2048
1647 SysSockGetLastErrorSync: 2048
1648 SysSockGetOption: 0
1649 SysSockHtonl: 0
1650 SysSockHtons: 0
1651 SysSockInetAddr: 0
1652 SysSockInetNtoa: 0
1653 SysSockIoctl: 0
1654 SysSockListen: 0
1655 SysSockNtohl: 0
1656 SysSockNtohs: 0
1657 SysSockRecv: 0
1658 SysSockRecvFrom: 0
1659 SysSockSelect: 0
1660 SysSockSend: 0
1661 SysSockSendTo: 0
1662 SysSockSetIPAddress: 0
1663 SysSockSetOption: 0
1664 SysSockShutdown: 0
1665 NumOfGVLs: 1
1666 Globale_Variablen: 0
1667 END_LIBRARY
1668
1669 LIBRARY
1670 Standard.lib 2.12.10 14:48:34
1671 (* @LIBRARYSYMFILEINFO := '0' *)
1672 NumOfPOUs: 26
1673 ASCIIBYTE_TO_STRING: 2048
1674 CONCAT: 0
1675 CTD: 0
1676 CTU: 0
1677 CTUD: 0
1678 DELETE: 0
1679 F_TRIG: 0
1680 FIND: 0
1681 INSERT: 0
1682 LEFT: 0
1683 LEN: 0
1684 MID: 0
1685 R_TRIG: 0
1686 REAL_STATE: 2048
1687 REPLACE: 0
1688 RIGHT: 0
1689 RS: 0
1690 RTC: 0
1691 SEMA: 0
1692 SR: 0
1693 STANDARD_VERSION: 2048
1694 STRING_COMPARE: 2048
1695 STRING_TO_ASCIIBYTE: 2048
1696 TOF: 0
1697 TON: 0
1698 TP: 0
```

```
1699 NumOfGVLs: 1
1700 'Global␣Variables␣0': 0
1701 END_LIBRARY
1702
1703 LIBRARY
1704 SYSLIBCALLBACK.LIB 2.12.10 14:48:32
1705 (* @LIBRARYSYMFILEINFO := '0' *)
1706 NumOfPOUs: 2
1707 SysCallbackRegister: 0
1708 SysCallbackUnregister: 0
1709 NumOfGVLs: 2
1710 Globale_Variablen: 0
1711 Version: 0
1712 END_LIBRARY
1713
1714 PLC_CONFIGURATION
1715 _GLOBAL
1716 _VERSION: 3
1717 _AUTOADR: 0
1718 _CHECKADR: 0
1719 _SAVECONFIGFILESINPROJECT: 0
1720 _END_GLOBAL
1721
1722 _MODULE: '3S'
1723 _SECTION_NAME: 'Root'
1724 _INDEX_IN_PARENT: '-1'
1725 _MODULE_NAME: 'Hardware␣configuration'
1726 _NODE_ID: -1
1727 _IECIN: %IB0
1728 _IECOUT: %QB0
1729 _IECDIAG: %MB0
1730 _DOWNLOAD: 1
1731 _EXCLUDEFROMAUTOADR: 0
1732 _COMMENT: ''
1733
1734 _MODULE: '3S'
1735 _SECTION_NAME: 'K_Bus'
1736 _INDEX_IN_PARENT: '1'
1737 _MODULE_NAME: 'K-Bus'
1738 _NODE_ID: 0
1739 _IECIN: %IB0
1740 _IECOUT: %QB0
1741 _IECDIAG: %MB0
1742 _DOWNLOAD: 1
1743 _EXCLUDEFROMAUTOADR: 0
1744 _COMMENT: 'Ethernet␣Controller␣100MBit␣2Port'
1745
1746 _MODULE: '3S'
1747 _SECTION_NAME: 'Type_14_2_Channels'
1748 _INDEX_IN_PARENT: '1'
1749 _MODULE_NAME: '0750-0554␣2␣AO␣4-20mA'
1750 _NODE_ID: 0
1751 _IECIN: %IB16
1752 _IECOUT: %QB16
1753 _IECDIAG: %MB0
1754 _DOWNLOAD: 1
1755 _EXCLUDEFROMAUTOADR: 0
1756 _COMMENT: ''
1757
1758 _PARAMETER
```

```
1759 _PARAM 10000: 0, 'PLC'
1760 _PARAM 10001: 0, 'plugged'
1761 _END_PARAMETER
1762
1763 _CHANNEL
1764 _SECTION_NAME: 'WORDOnW_Q'
1765 _INDEX_IN_PARENT: '1'
1766 _SYMBOLIC_NAME: 'MV101_clamp'
1767 _COMMENT: 'Ch_1␣Analog␣output'
1768 _CHANNEL_MODE: 'Q'
1769 _IECADR: %QW0
1770 _END_CHANNEL
1771
1772 _CHANNEL
1773 _SECTION_NAME: 'WORDOnW_Q'
1774 _INDEX_IN_PARENT: '2'
1775 _SYMBOLIC_NAME: 'MV301_clamp'
1776 _COMMENT: 'Ch_2␣Analog␣output'
1777 _CHANNEL_MODE: 'Q'
1778 _IECADR: %QW1
1779 _END_CHANNEL
1780 _END_MODULE
1781
1782 _MODULE: '3S'
1783 _SECTION_NAME: 'Type_1_2_Channels'
1784 _INDEX_IN_PARENT: '2'
1785 _MODULE_NAME: '0750−0400␣2␣DI␣24␣V␣DC␣3.0ms'
1786 _NODE_ID: 1
1787 _IECIN: %IB24
1788 _IECOUT: %QB16
1789 _IECDIAG: %MB0
1790 _DOWNLOAD: 1
1791 _EXCLUDEFROMAUTOADR: 0
1792 _COMMENT: ''
1793
1794 _PARAMETER
1795 _PARAM 10000: 0, 'PLC'
1796 _PARAM 10001: 0, 'plugged'
1797 _END_PARAMETER
1798
1799 _CHANNEL
1800 _SECTION_NAME: 'BOOLOnX_I'
1801 _INDEX_IN_PARENT: '1'
1802 _SYMBOLIC_NAME: 'TS_clamp'
1803 _COMMENT: 'Ch_1␣Digital␣input'
1804 _CHANNEL_MODE: 'I'
1805 _IECADR: %IX12.0
1806 _END_CHANNEL
1807
1808 _CHANNEL
1809 _SECTION_NAME: 'BOOLOnX_I'
1810 _INDEX_IN_PARENT: '2'
1811 _SYMBOLIC_NAME: 'TZ_clamp'
1812 _COMMENT: 'Ch_2␣Digital␣input'
1813 _CHANNEL_MODE: 'I'
1814 _IECADR: %IX12.1
1815 _END_CHANNEL
1816 _END_MODULE
1817
1818 _MODULE: '3S'
```

```
1819 _SECTION_NAME: 'Type_44_4_Channels'
1820 _INDEX_IN_PARENT: '3'
1821 _MODULE_NAME: '0750-0460/0000-0003 4 AI PT1000 (RTD)'
1822 _NODE_ID: 2
1823 _IECIN: %IB16
1824 _IECOUT: %QB16
1825 _IECDIAG: %MB0
1826 _DOWNLOAD: 1
1827 _EXCLUDEFROMAUTOADR: 0
1828 _COMMENT: ''
1829
1830 _PARAMETER
1831 _PARAM 10000: 0, 'PLC'
1832 _PARAM 10001: 0, 'plugged'
1833 _END_PARAMETER
1834
1835 _CHANNEL
1836 _SECTION_NAME: 'INTOnW_I'
1837 _INDEX_IN_PARENT: '1'
1838 _SYMBOLIC_NAME: 'TIRC103_clamp'
1839 _COMMENT: 'Ch_1 signed Input value'
1840 _CHANNEL_MODE: 'I'
1841 _IECADR: %IW0
1842 _END_CHANNEL
1843
1844 _CHANNEL
1845 _SECTION_NAME: 'INTOnW_I'
1846 _INDEX_IN_PARENT: '2'
1847 _SYMBOLIC_NAME: 'TC101_clamp'
1848 _COMMENT: 'Ch_2 signed Input value'
1849 _CHANNEL_MODE: 'I'
1850 _IECADR: %IW1
1851 _END_CHANNEL
1852
1853 _CHANNEL
1854 _SECTION_NAME: 'INTOnW_I'
1855 _INDEX_IN_PARENT: '3'
1856 _SYMBOLIC_NAME: 'TIRC305_clamp'
1857 _COMMENT: 'Ch_3 signed Input value'
1858 _CHANNEL_MODE: 'I'
1859 _IECADR: %IW2
1860 _END_CHANNEL
1861
1862 _CHANNEL
1863 _SECTION_NAME: 'INTOnW_I'
1864 _INDEX_IN_PARENT: '4'
1865 _SYMBOLIC_NAME: 'TIRC102_clamp'
1866 _COMMENT: 'Ch_4 signed Input value'
1867 _CHANNEL_MODE: 'I'
1868 _IECADR: %IW3
1869 _END_CHANNEL
1870 _END_MODULE
1871
1872 _MODULE: '3S'
1873 _SECTION_NAME: 'Type_14_2_Channels'
1874 _INDEX_IN_PARENT: '4'
1875 _MODULE_NAME: '0750-0554 2 AO 4-20mA'
1876 _NODE_ID: 3
1877 _IECIN: %IB16
1878 _IECOUT: %QB20
```

```
1879 _IECDIAG: %MB0
1880 _DOWNLOAD: 1
1881 _EXCLUDEFROMAUTOADR: 0
1882 _COMMENT: ''
1883
1884 _PARAMETER
1885 _PARAM 10000: 0, 'PLC'
1886 _PARAM 10001: 0, 'plugged'
1887 _END_PARAMETER
1888
1889 _CHANNEL
1890 _SECTION_NAME: 'WORDOnW_Q'
1891 _INDEX_IN_PARENT: '1'
1892 _SYMBOLIC_NAME: 'MV302_clamp'
1893 _COMMENT: 'Ch_1_Analog_output'
1894 _CHANNEL_MODE: 'Q'
1895 _IECADR: %QW2
1896 _END_CHANNEL
1897
1898 _CHANNEL
1899 _SECTION_NAME: 'WORDOnW_Q'
1900 _INDEX_IN_PARENT: '2'
1901 _SYMBOLIC_NAME: ''
1902 _COMMENT: 'Ch_2_Analog_output'
1903 _CHANNEL_MODE: 'Q'
1904 _IECADR: %QW3
1905 _END_CHANNEL
1906 _END_MODULE
1907
1908 _MODULE: '3S'
1909 _SECTION_NAME: 'Type_14_4_Channels'
1910 _INDEX_IN_PARENT: '5'
1911 _MODULE_NAME: '0750-0559_4_AO_0-10V_DC'
1912 _NODE_ID: 4
1913 _IECIN: %IB24
1914 _IECOUT: %QB20
1915 _IECDIAG: %MB0
1916 _DOWNLOAD: 1
1917 _EXCLUDEFROMAUTOADR: 0
1918 _COMMENT: ''
1919
1920 _PARAMETER
1921 _PARAM 10000: 0, 'PLC'
1922 _PARAM 10001: 0, 'plugged'
1923 _END_PARAMETER
1924
1925 _CHANNEL
1926 _SECTION_NAME: 'WORDOnW_Q'
1927 _INDEX_IN_PARENT: '1'
1928 _SYMBOLIC_NAME: 'P401_clamp'
1929 _COMMENT: 'Ch_1_Analog_output'
1930 _CHANNEL_MODE: 'Q'
1931 _IECADR: %QW4
1932 _END_CHANNEL
1933
1934 _CHANNEL
1935 _SECTION_NAME: 'WORDOnW_Q'
1936 _INDEX_IN_PARENT: '2'
1937 _SYMBOLIC_NAME: ''
1938 _COMMENT: 'Ch_2_Analog_output'
```

```
1939 _CHANNEL_MODE: 'Q'
1940 _IECADR: %QW5
1941 _END_CHANNEL
1942
1943 _CHANNEL
1944 _SECTION_NAME: 'WORDOnW_Q'
1945 _INDEX_IN_PARENT: '3'
1946 _SYMBOLIC_NAME: ''
1947 _COMMENT: 'Ch_3␣Analog␣output'
1948 _CHANNEL_MODE: 'Q'
1949 _IECADR: %QW6
1950 _END_CHANNEL
1951
1952 _CHANNEL
1953 _SECTION_NAME: 'WORDOnW_Q'
1954 _INDEX_IN_PARENT: '4'
1955 _SYMBOLIC_NAME: ''
1956 _COMMENT: 'Ch_4␣Analog␣output'
1957 _CHANNEL_MODE: 'Q'
1958 _IECADR: %QW7
1959 _END_CHANNEL
1960 _END_MODULE
1961
1962 _MODULE: '3S'
1963 _SECTION_NAME: 'Type_44_4_Channels'
1964 _INDEX_IN_PARENT: '6'
1965 _MODULE_NAME: '0750−0460/0000−0003␣4␣AI␣PT1000␣(RTD)'
1966 _NODE_ID: 5
1967 _IECIN: %IB24
1968 _IECOUT: %QB24
1969 _IECDIAG: %MB0
1970 _DOWNLOAD: 1
1971 _EXCLUDEFROMAUTOADR: 0
1972 _COMMENT: ''
1973
1974 _PARAMETER
1975 _PARAM 10000: 0, 'PLC'
1976 _PARAM 10001: 0, 'plugged'
1977 _END_PARAMETER
1978
1979 _CHANNEL
1980 _SECTION_NAME: 'INTOnW_I'
1981 _INDEX_IN_PARENT: '1'
1982 _SYMBOLIC_NAME: 'TIR301_clamp'
1983 _COMMENT: 'Ch_1␣signed␣Input␣value'
1984 _CHANNEL_MODE: 'I'
1985 _IECADR: %IW4
1986 _END_CHANNEL
1987
1988 _CHANNEL
1989 _SECTION_NAME: 'INTOnW_I'
1990 _INDEX_IN_PARENT: '2'
1991 _SYMBOLIC_NAME: 'storage2_TIRC320'
1992 _COMMENT: 'Ch_2␣signed␣Input␣value'
1993 _CHANNEL_MODE: 'I'
1994 _IECADR: %IW5
1995 _END_CHANNEL
1996
1997 _CHANNEL
1998 _SECTION_NAME: 'INTOnW_I'
```

```
1999 _INDEX_IN_PARENT: '3'
2000 _SYMBOLIC_NAME: 'TIRC302_clamp'
2001 _COMMENT: 'Ch_3␣signed␣Input␣value'
2002 _CHANNEL_MODE: 'I'
2003 _IECADR: %IW6
2004 _END_CHANNEL
2005
2006 _CHANNEL
2007 _SECTION_NAME: 'INTOnW_I'
2008 _INDEX_IN_PARENT: '4'
2009 _SYMBOLIC_NAME: 'TIRC401_clamp'
2010 _COMMENT: 'Ch_4␣signed␣Input␣value'
2011 _CHANNEL_MODE: 'I'
2012 _IECADR: %IW7
2013 _END_CHANNEL
2014 _END_MODULE
2015
2016 _MODULE: '3S'
2017 _SECTION_NAME: 'Type_44_4_Channels'
2018 _INDEX_IN_PARENT: '7'
2019 _MODULE_NAME: '0750−0460/0000−0003␣4␣AI␣PT1000␣(RTD)'
2020 _NODE_ID: 6
2021 _IECIN: %IB24
2022 _IECOUT: %QB32
2023 _IECDIAG: %MB0
2024 _DOWNLOAD: 1
2025 _EXCLUDEFROMAUTOADR: 0
2026 _COMMENT: ''
2027
2028 _PARAMETER
2029 _PARAM 10000: 0, 'PLC'
2030 _PARAM 10001: 0, 'plugged'
2031 _END_PARAMETER
2032
2033 _CHANNEL
2034 _SECTION_NAME: 'INTOnW_I'
2035 _INDEX_IN_PARENT: '1'
2036 _SYMBOLIC_NAME: 'storage1_TIRC322'
2037 _COMMENT: 'Ch_1␣signed␣Input␣value'
2038 _CHANNEL_MODE: 'I'
2039 _IECADR: %IW8
2040 _END_CHANNEL
2041
2042 _CHANNEL
2043 _SECTION_NAME: 'INTOnW_I'
2044 _INDEX_IN_PARENT: '2'
2045 _SYMBOLIC_NAME: 'storage1_TIRC324'
2046 _COMMENT: 'Ch_2␣signed␣Input␣value'
2047 _CHANNEL_MODE: 'I'
2048 _IECADR: %IW9
2049 _END_CHANNEL
2050
2051 _CHANNEL
2052 _SECTION_NAME: 'INTOnW_I'
2053 _INDEX_IN_PARENT: '3'
2054 _SYMBOLIC_NAME: 'storage2_TIRC321'
2055 _COMMENT: 'Ch_3␣signed␣Input␣value'
2056 _CHANNEL_MODE: 'I'
2057 _IECADR: %IW10
2058 _END_CHANNEL
```

```
2059
2060 _CHANNEL
2061 _SECTION_NAME: 'INTOnW_I'
2062 _INDEX_IN_PARENT: '4'
2063 _SYMBOLIC_NAME: 'storage1_TIRC323'
2064 _COMMENT: 'Ch_4_signed_Input_value'
2065 _CHANNEL_MODE: 'I'
2066 _IECADR: %IW11
2067 _END_CHANNEL
2068 _END_MODULE
2069 _END_MODULE
2070
2071 _MODULE: '3S'
2072 _SECTION_NAME: 'FB_VARS'
2073 _INDEX_IN_PARENT: '2'
2074 _MODULE_NAME: 'Fieldbus_variables'
2075 _NODE_ID: 1
2076 _IECIN: %IB0
2077 _IECOUT: %QB0
2078 _IECDIAG: %MB0
2079 _DOWNLOAD: 1
2080 _EXCLUDEFROMAUTOADR: 0
2081 _COMMENT: ''
2082 _END_MODULE
2083
2084 _MODULE: '3S'
2085 _SECTION_NAME: 'FLAG_VARS'
2086 _INDEX_IN_PARENT: '3'
2087 _MODULE_NAME: 'Flag_variables'
2088 _NODE_ID: 2
2089 _IECIN: %IB0
2090 _IECOUT: %QB0
2091 _IECDIAG: %MB0
2092 _DOWNLOAD: 1
2093 _EXCLUDEFROMAUTOADR: 0
2094 _COMMENT: ''
2095 _END_MODULE
2096
2097 _MODULE: '3S'
2098 _SECTION_NAME: 'MB_MASTER'
2099 _INDEX_IN_PARENT: '4'
2100 _MODULE_NAME: 'Modbus-Master'
2101 _NODE_ID: 3
2102 _IECIN: %IB0
2103 _IECOUT: %QB0
2104 _IECDIAG: %MB0
2105 _DOWNLOAD: 1
2106 _EXCLUDEFROMAUTOADR: 0
2107 _COMMENT: ''
2108 _END_MODULE
2109 _END_MODULE
2110 PLC_END
2111
2112
2113 RESOURCE
2114
2115 TASK TaskTWW (PRIORITY := 1, INTERVAL := T#200ms);
2116 PLC_PRG_TWW();
2117 {Additional_info : 1,0,0,0,1,4294967295}
2118 END_TASK
```

```
2119
2120 TASK TaskPID (PRIORITY := 3, INTERVAL := T#200ms);
2121 PID_generator();
2122 PID_MVdreinulleins();
2123 PID_MVeinsnulleins();
2124 PID_Pviernulleins();
2125 {Additional_info : 1,0,0,0,1,4294967295}
2126 END_TASK
2127
2128 TASK TaskWUEST (PRIORITY := 2, INTERVAL := T#200ms);
2129 PLC_PRG_WUEST();
2130 {Additional_info : 1,0,0,0,1,4294967295}
2131 END_TASK
2132
2133 TASK TaskSETOUTPUT (PRIORITY := 4, INTERVAL := T#200ms);
2134 SETOUTPUT();
2135 {Additional_info : 1,0,0,0,1,4294967295}
2136 END_TASK
2137 {event_task : 'start','Called␣when␣program␣starts','','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT
2138 ␣␣␣␣␣␣␣␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;␣dwOwner:␣DWORD;␣END_VAR␣'}{event_task_info : 0,1,11988}
2139 {event_task : 'stop','Called␣when␣program␣stops','','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT
2140 ␣␣␣␣␣␣␣␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;␣dwOwner:␣DWORD;␣END_VAR␣'}{event_task_info : 0,2,11988}
2141 {event_task : 'before_reset','Called␣before␣reset␣takes␣place','','FUNCTION␣systemevent:
2142 ␣␣␣␣␣␣␣␣␣␣DWORD␣VAR_INPUT␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;␣dwOwner:␣DWORD;␣END_VAR␣'}
2143 {event_task_info : 0,3,11988}
2144 {event_task : 'after_reset','Called␣after␣reset␣took␣place','','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT
2145 ␣␣␣␣␣␣␣␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;␣dwOwner:␣DWORD;␣END_VAR␣'}{event_task_info : 0,4,11988}
2146 {event_task : 'shutdown','Called␣before␣shutdown␣is␣performed␣(Firmware␣update␣over␣ethernet)','
2147 ␣␣␣␣␣␣␣␣','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;
2148 ␣␣␣␣␣␣␣␣dwOwner:␣DWORD;␣END_VAR␣'}{event_task_info : 0,5,11988}
2149 {event_task : 'excpt_watchdog','Software␣watchdog␣of␣IEC-task␣expired','
2150 ␣␣␣␣␣␣␣␣','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;
2151 ␣␣␣␣␣␣␣␣dwOwner:␣DWORD;␣END_VAR␣'}{event_task_info : 0,7,11988}
2152 {event_task : 'excpt_fieldbus','Fieldbus␣error','','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT
2153 ␣␣␣␣␣␣␣␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;␣dwOwner:␣DWORD;␣END_VAR␣'}{event_task_info : 0,9,11988}
2154 {event_task : 'excpt_ioupdate','KBus␣error','','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT
2155 ␣␣␣␣␣␣␣␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;␣dwOwner:␣DWORD;␣END_VAR␣'}{event_task_info : 0,10,11988}
2156 {event_task : 'excpt_dividebyzero','Division␣by␣zero.␣Only␣integer␣operations!','
2157 ␣␣␣␣␣␣␣␣','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;
2158 ␣␣␣␣␣␣␣␣dwOwner:␣DWORD;␣END_VAR␣'}{event_task_info : 0,18,11988}
2159 {event_task : 'excpt_noncontinuable','Exception␣handler','','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT
2160 ␣␣␣␣␣␣␣␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;␣dwOwner:␣DWORD;␣END_VAR␣'}{event_task_info : 0,20,11988}
2161 {event_task : 'after_reading_inputs','Called␣after␣reading␣of␣inputs','
2162 ␣␣␣␣␣␣␣␣','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;
2163 ␣␣␣␣␣␣␣␣dwOwner:␣DWORD;␣END_VAR␣'}{event_task_info : 0,28,11988}
2164 {event_task : 'before_writing_outputs','Called␣before␣writing␣of␣outputs','
2165 ␣␣␣␣␣␣␣␣','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;
2166 ␣␣␣␣␣␣␣␣dwOwner:␣DWORD;␣END_VAR␣'}{event_task_info : 0,29,11988}
2167 {event_task : 'debug_loop','Debug␣loop␣at␣breakpoint','','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT
2168 ␣␣␣␣␣␣␣␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;␣dwOwner:␣DWORD;␣END_VAR␣'}{event_task_info : 0,31,11988}
2169 {event_task : 'online_change','Is␣called␣after␣CodeInit()␣at␣Online-Change','
2170 ␣␣␣␣␣␣␣␣','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;
2171 ␣␣␣␣␣␣␣␣dwOwner:␣DWORD;␣END_VAR␣'}{event_task_info : 0,33,11988}
2172 {event_task : 'before_download','Is␣called␣before␣the␣Download␣starts','
2173 ␣␣␣␣␣␣␣␣','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;␣dwOwner:␣DWORD;
2174 ␣␣␣␣␣␣␣␣END_VAR␣'}{event_task_info : 0,34,11988}
2175 {event_task : 'event_login','Is␣called␣before␣the␣login␣service␣is␣performed','
2176 ␣␣␣␣␣␣␣␣','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;
2177 ␣␣␣␣␣␣␣␣dwOwner:␣DWORD;␣END_VAR␣'}{event_task_info : 0,501,11988}
2178 {event_task : 'eth_overload','Ethernet␣Overload','','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT
```

```
2179 ␣␣␣␣␣␣␣␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;␣dwOwner:␣DWORD;␣END_VAR␣'}{event_task_info : 0,750,11988}
2180 {event_task : 'eth_network_ready','Is␣called␣directly␣after␣the␣Network␣and␣the␣PLC␣are␣initialised','
2181 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;
2182 ␣␣␣␣␣␣␣␣dwOwner:␣DWORD;␣END_VAR␣'}{event_task_info : 0,751,11988}
2183 {event_task : 'blink_code','New␣blink␣code␣/␣Blink␣code␣cleared␣(␣Call␣STATUS_GET_LAST_ERROR
2184 ␣␣␣␣␣␣␣␣␣␣for␣details␣)','','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;
2185 ␣␣␣␣␣␣␣␣dwOwner:␣DWORD;␣END_VAR␣'}{event_task_info : 0,752,11988}
2186 {event_task : 'interrupt_0','Interrupt␣Real␣Time␣Clock␣(every␣second)','
2187 ␣␣␣␣␣␣␣␣','FUNCTION␣systemevent:␣DWORD␣VAR_INPUT␣dwEvent:␣DWORD;␣dwFilter:␣DWORD;␣dwOwner:␣DWORD;
2188 ␣␣␣␣␣␣␣␣END_VAR␣'}{event_task_info : 0,1000,11988}
2189
2190 END_RESOURCE
```

Bestelllisten für die Hardwaretestumgebung

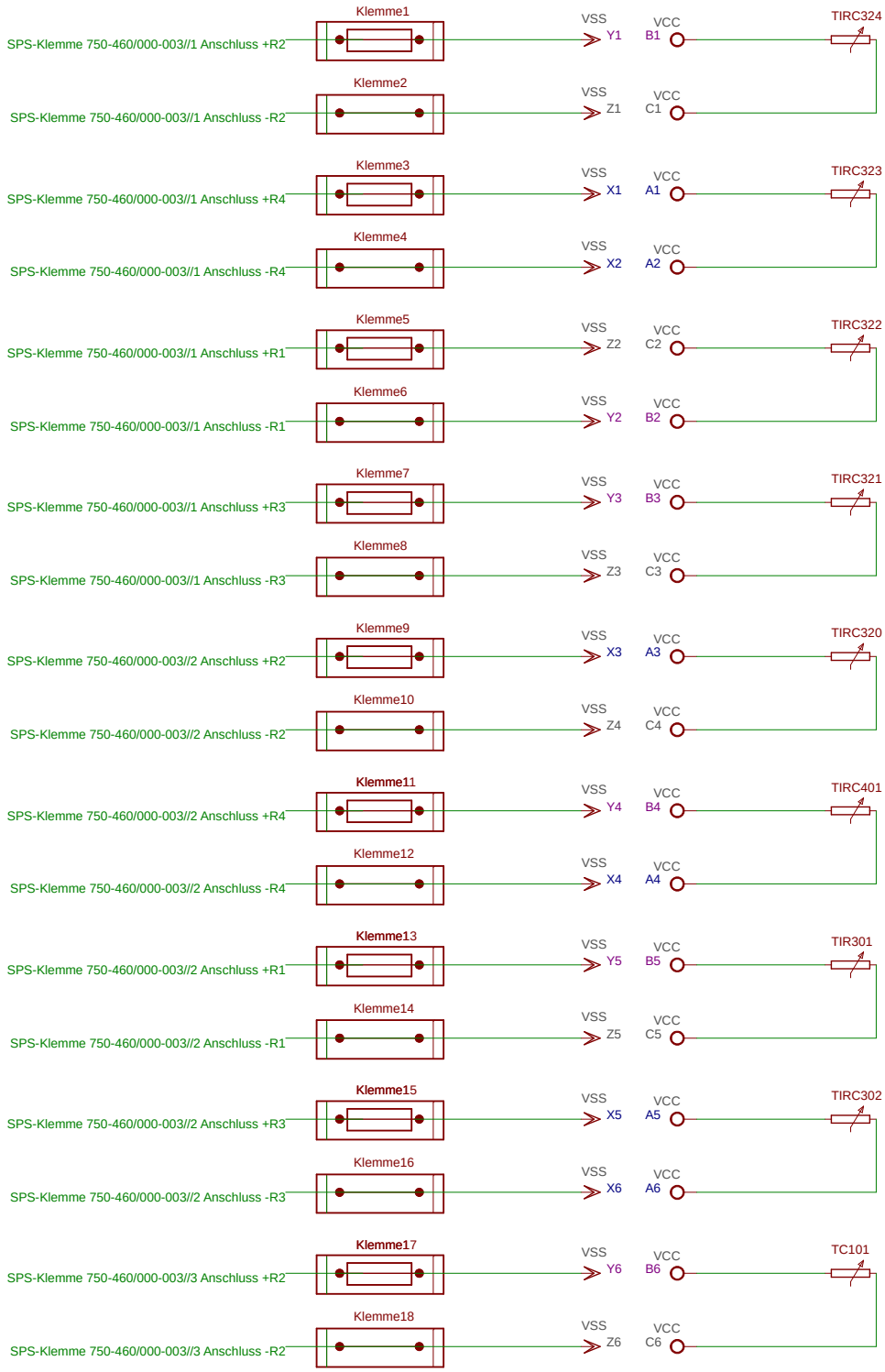
Pos.	Artikel	Menge	Einzelpreis netto	Ust-Satz in %	MwSt. in €	Gesamt brutto in €
1	Schaltdraht 1m (schwarz) 1m Bestell-Nr.: 607051 - 62	20	0,50 €	19%	0,09	11,80
2	Schaltdraht 1m (blau) 1m Bestell-Nr.: 607077 - 62	20	0,53 €	19%	0,10	12,60
3	Potentiometer 0 - 1 kOhm Bestell-Nr.: 429260 - 62	25	12,93 €	19%	2,46	384,75
4	Drehknopf für Potentiometer Bestell-Nr.: 429924 - 62	25	1,23 €	19%	0,23	36,50
5	Kippschalter Bestell-Nr.: 701147 - 62	5	2,26 €	19%	0,43	13,45
6	LED inkl. Fassung blau Bestell-Nr.: 1605949 - 62	3	3,32 €	19%	0,63	11,85
7	Installations-Gehäuse AKL 2-th Bestell-Nr.: 531129 - 62	1	97,48 €	19%	18,52	116,00
8	Hutschiene Bestell-Nr.: 531167 - 62	3	4,82 €	19%	0,92	17,22
9	Trennklemmen Bestell-Nr.: 733167 - 62	22	1,66 €	19%	0,31	43,34
10	Abschlussdeckel Trennklemme Bestell-Nr.: 733205 - 62	4	0,41 €	19%	0,08	1,96
11	Bauelemente-Stecker Bestell-Nr.: 744059 - 62	22	3,06 €	19%	0,58	80,08
12	Abteilungstrennplatte Bestell-Nr.: 744559 - 62	4	0,34 €	19%	0,07	1,64
13	Durchgangsklemmen Bestell-Nr.: 587995 - 62	1	31,88 €	19%	6,06	37,94
Liefer- und Versandkosten						
		Summe				769,13

1	DC Drehspul Amperemeter Einbaumessgerät, analog, 4 → 20mA Art.Nr.: 886-1973	4	22,39 €	19%	4,25	106,58
Liefer- und Versandkosten						106,58

Pos.	Artikel	Menge	Einzelpreis netto	Ust-Satz in %	MwSt. in €	Gesamt brutto in €
1	Bananenbuchse 4mm, vollisoliert, blau, Art.-Nr: BB 4 BL BD	14	0,24 €	19%	0,06	4,20
2	Bananenbuchse 4mm, vollisoliert, schwarz, Art.-Nr: BB 4 SW BD	14	0,26 €	19%	0,06	4,48
3	Bananenbuchse 4mm, vollisoliert, rot, Art.-Nr: BB4 RT BD	14	0,26 €	19%	0,06	4,48
4	Bananenstecker, 4mm, schwarz, Art.-Nr: BKL 072150-P	14	0,81 €	19%	0,19	14,00
5	Bananenstecker, 4mm, rot, , Art.-Nr: BKL 072149-P	14	0,81 €	19%	0,19	14,00
6	Bananenstecker, 4mm, blau, , Art.-Nr: BKL 072153-P	14	0,81 €	19%	0,19	13,86
7	Versandkosten					5,60
		Summe				60,62

Abbildung A.4.: Bestelllisten der Hardwaretestumgebung

Schaltplan und Erklärung


Klemme1
SPS-Klemme 750-460/000-003//1 Anschluss +R2
VSS
Y1
VCC
B1
TIRC324
Klemme2
SPS-Klemme 750-460/000-003//1 Anschluss -R2
VSS
Z1
VCC
C1
Klemme3
SPS-Klemme 750-460/000-003//1 Anschluss +R4
VSS
X1
VCC
A1
TIRC323
Klemme4
SPS-Klemme 750-460/000-003//1 Anschluss -R4
VSS
X2
VCC
A2
Klemme5
SPS-Klemme 750-460/000-003//1 Anschluss +R1
VSS
Z2
VCC
C2
TIRC322
Klemme6
SPS-Klemme 750-460/000-003//1 Anschluss -R1
VSS
Y2
VCC
B2
Klemme7
SPS-Klemme 750-460/000-003//1 Anschluss +R3
VSS
Y3
VCC
B3
TIRC321
Klemme8
SPS-Klemme 750-460/000-003//1 Anschluss -R3
VSS
Z3
VCC
C3
Klemme9
SPS-Klemme 750-460/000-003//2 Anschluss +R2
VSS
X3
VCC
A3
TIRC320
Klemme10
SPS-Klemme 750-460/000-003//2 Anschluss -R2
VSS
Z4
VCC
C4
Klemme11
SPS-Klemme 750-460/000-003//2 Anschluss +R4
VSS
Y4
VCC
B4
TIRC401
Klemme12
SPS-Klemme 750-460/000-003//2 Anschluss -R4
VSS
X4
VCC
A4
Klemme13
SPS-Klemme 750-460/000-003//2 Anschluss +R1
VSS
Y5
VCC
B5
TIR301
Klemme14
SPS-Klemme 750-460/000-003//2 Anschluss -R1
VSS
Z5
VCC
C5
Klemme15
SPS-Klemme 750-460/000-003//2 Anschluss +R3
VSS
X5
VCC
A5
TIRC302
Klemme16
SPS-Klemme 750-460/000-003//2 Anschluss -R3
VSS
X6
VCC
A6
Klemme17
SPS-Klemme 750-460/000-003//3 Anschluss +R2
VSS
Y6
VCC
B6
TC101
Klemme18
SPS-Klemme 750-460/000-003//3 Anschluss -R2
VSS
Z6
VCC
C6

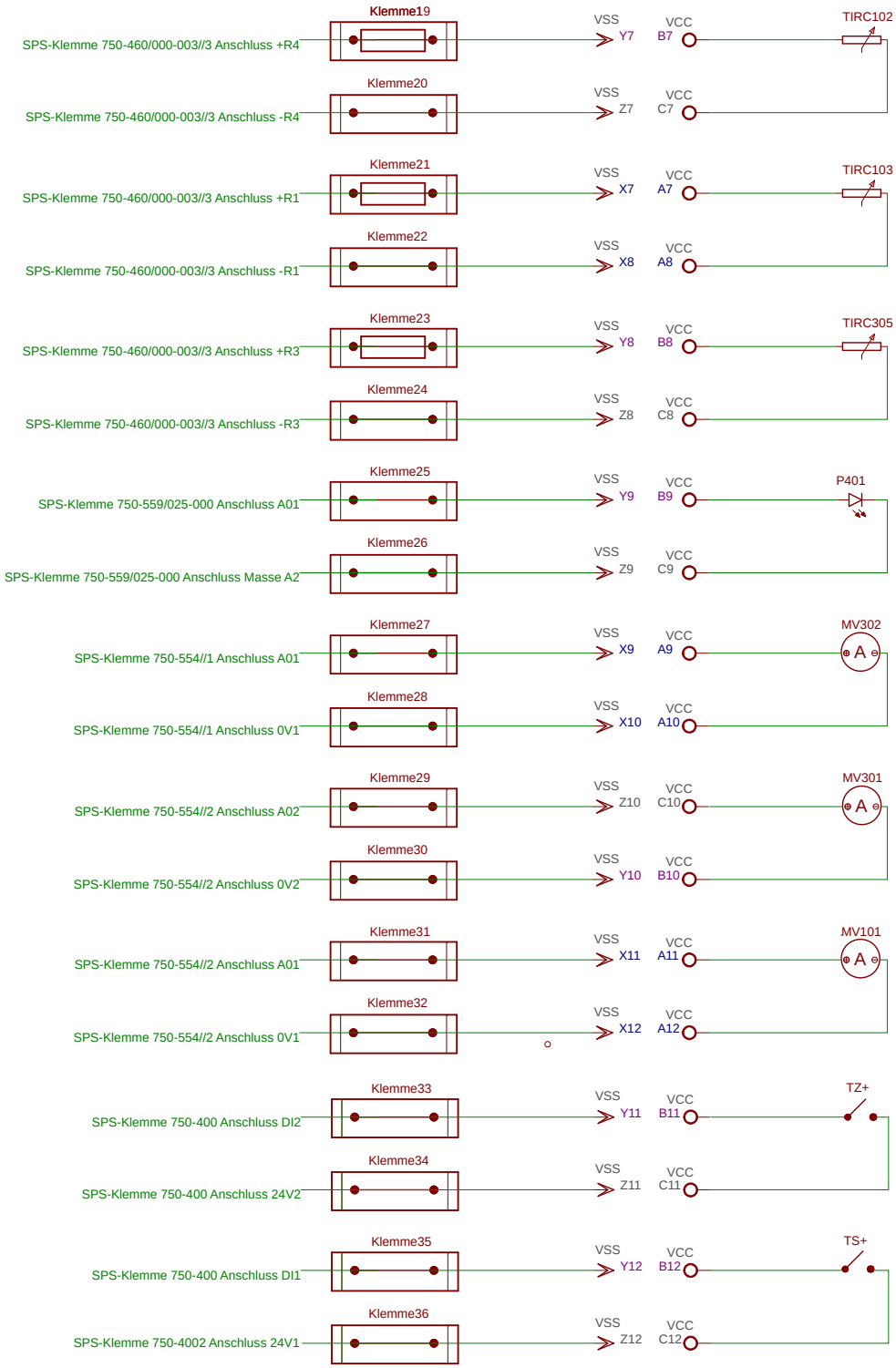


Abbildung A.5.: Schaltplan der Hardwaretestumgebung

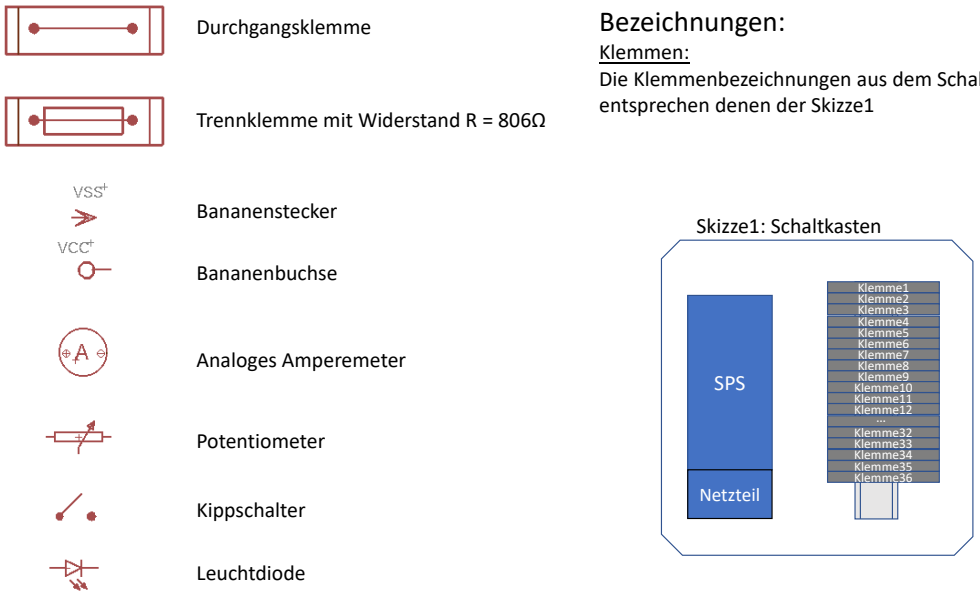
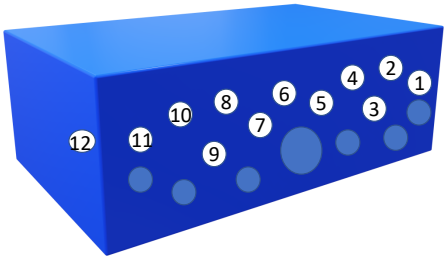


Abbildung A.6.: Legende der Schaltzeichen, Erklärung zur Klemmenkennzeichnung des Schaltplans

Bezeichnungen:
Bananenstecker:
Die Bananenstecker sind bezüglich des Lochs aus Skizze2, welche den Schaltkasten mit den Öffnungen zeigt, und der Farbe des Steckers in dem Schaltbild gekennzeichnet. Dabei gilt folgende Farbkennung:
Blauer Stecker: X
Roter Stecker: Y
Schwarzer Stecker: Z
Dementsprechend ist bspw. der blaue Stecker aus Öffnung mit X1 gekennzeichnet, der rote Stecker aus Öffnung 10 mit Y10.



Skizze2: Öffnungen des Schaltkastens

Abbildung A.7.: Kennzeichnung der Bananenstecker

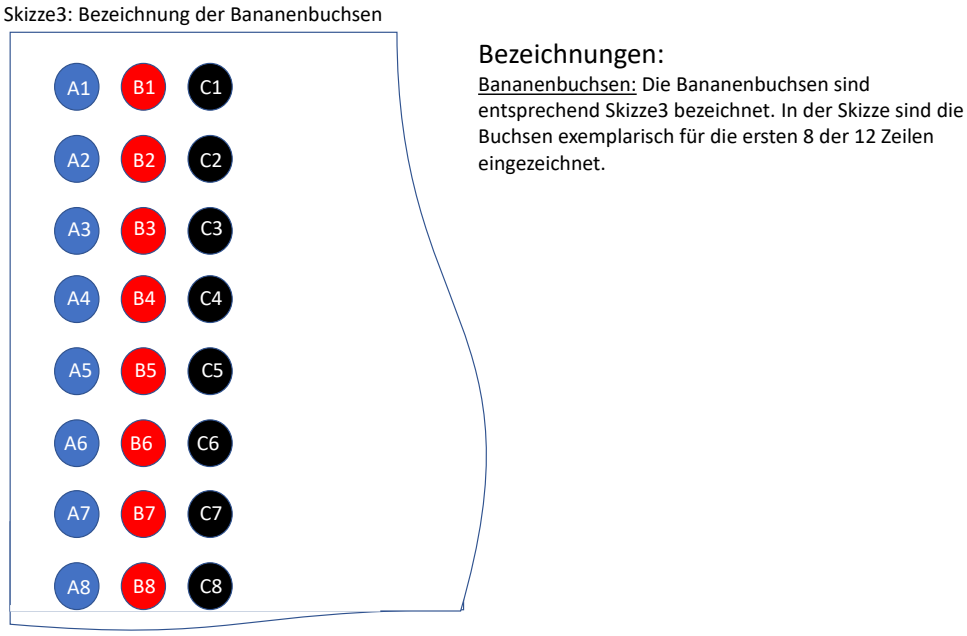


Abbildung A.8.: Kennzeichnung der Bananenbuchsen

Maße der Acrylglasplatte

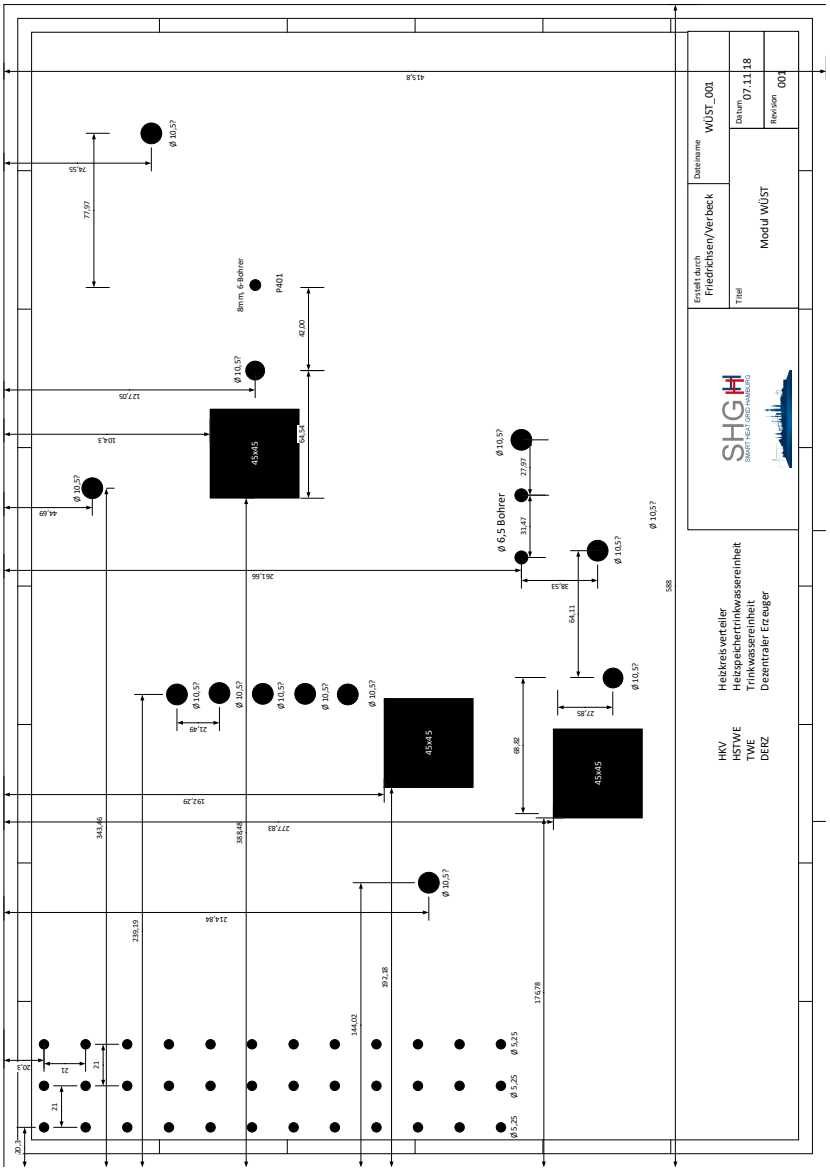


Abbildung A.9.: Maße für Zuschnitt der Acrylglasplatte

Kennlinien zur Einstellung der Regelparameter

Abbildung A.10.: Selbstentladung *storage1*

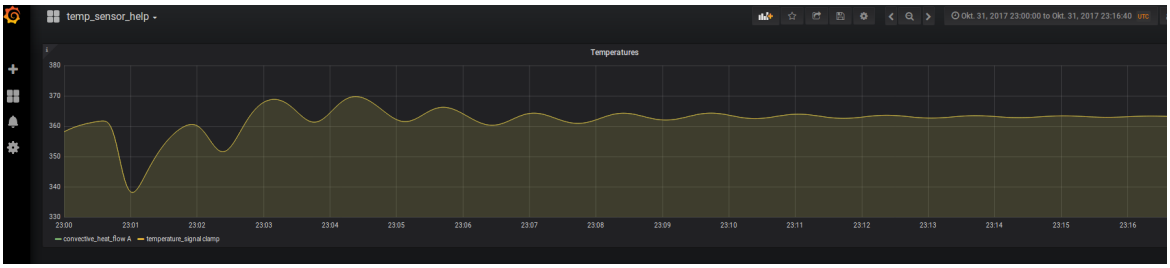


Abbildung A.11.: Parametrierung PID_generator (temp_sensor_help): Simulierte Temperatur temp_sensor_help für Kp = 0,000005, Tn = unendlich, default y von Analog Drive Controll = 0.004

Abbildung A.12.: Parametrierung PID_MV101: Simulierte Temperatur und Massenstrom der Rohrleitung pipe1 vor der Pumpe MV101

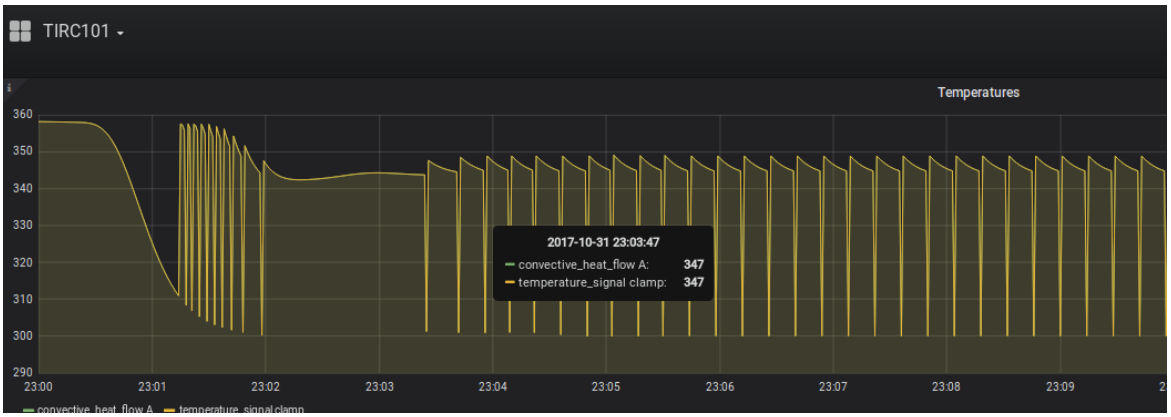


Abbildung A.13.: Parametrierung PID_MV101: Simulierte Temperatur des Temperatursensors TC101

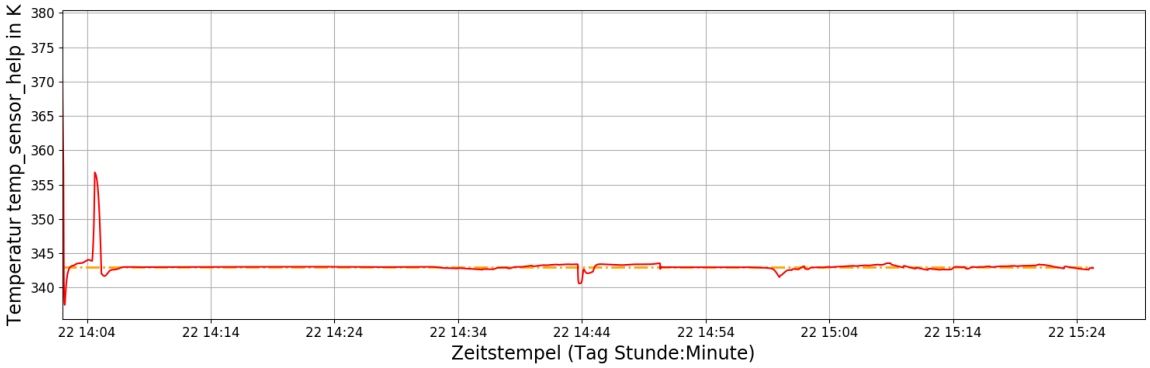


Abbildung A.14.: Parametrierung PID_MV301: Temperatur des Temperatursensors temp_sensor_help

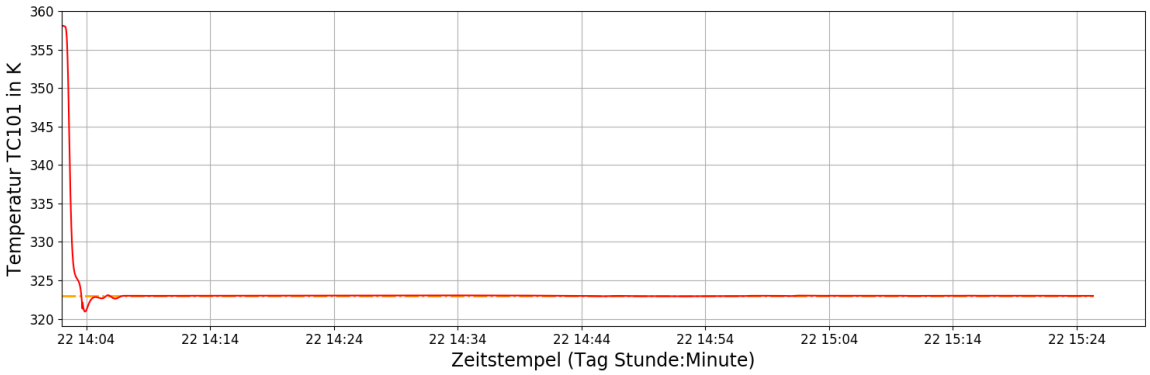


Abbildung A.15.: Temperatur des Temperatursensors TC101

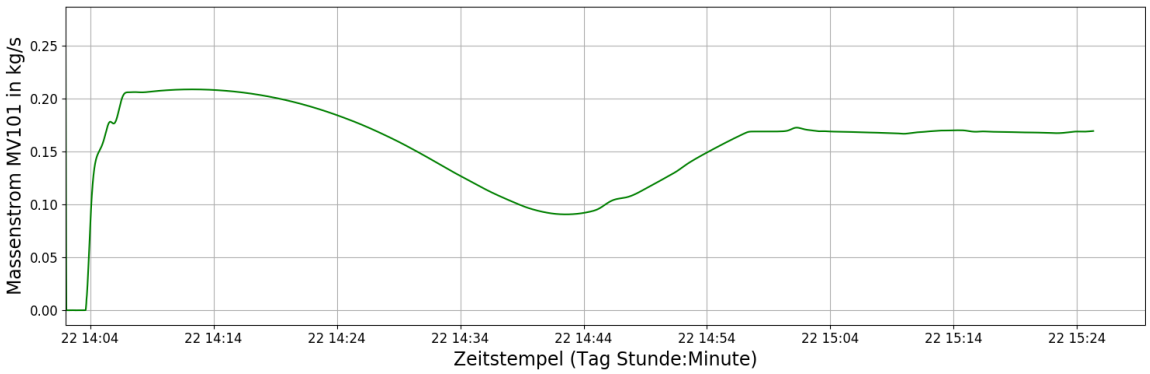


Abbildung A.16.: Massenstrom der Pumpe MV101

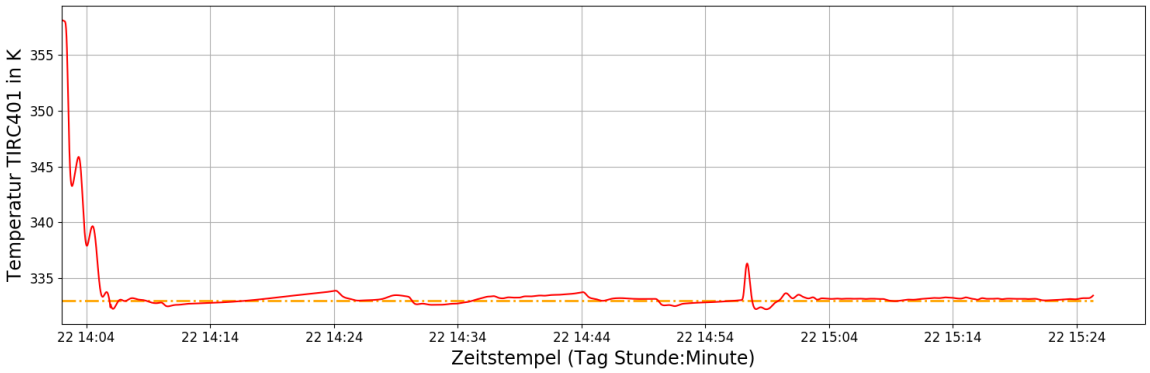


Abbildung A.17.: Temperatur des Temperatursensors TIRC401

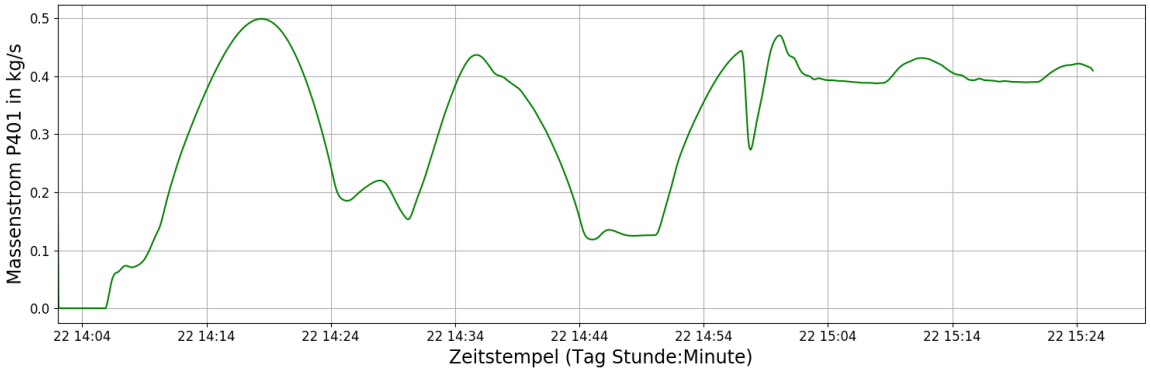
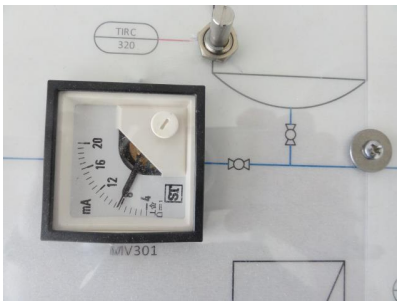


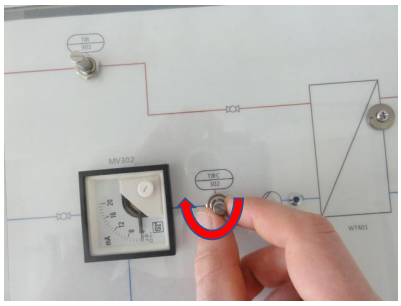
Abbildung A.18.: Massenstrom der Pumpe P401



TIRC302_clamp_ = 48 TIRC302_clamp_max = 55
Szenario_TWW = 10
SOC_soll = 80 SOC_RL_Begrenzung = 10
FLAG_9_TWW = FALSE

TIRC302_clamp_ = 48 TIRC302_clamp_zul... = 50
FLAG_9_TWW = FALSE

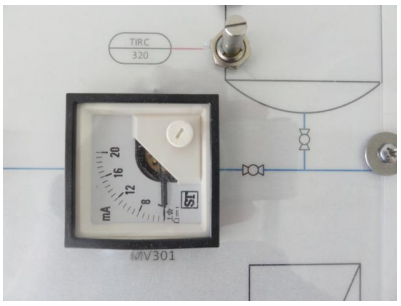
(1)Normalbetrieb: Ventil ca. ¼ geöffnet
SOC_soll = SOC_Leitwarte = 80%



TIRC302_clamp_ = 55.4 TIRC302_clamp_max = 55
Szenario_TWW = 9
SOC_soll = 10 SOC_RL_Begrenzung = 10
FLAG_9_TWW = TRUE

TIRC302_clamp_ = 55.4 TIRC302_clamp_zul... = 50
FLAG_9_TWW = TRUE

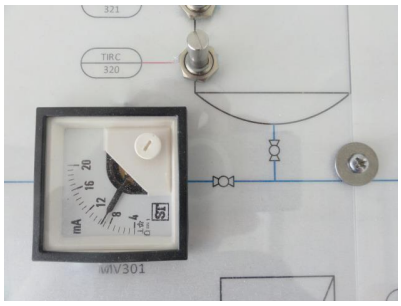
(2)Rücklauftemperaturbegrenzung aktiviert durch Erhöhen des Widerstands TIRC302, sodass
TIRC302_clamp_ > TIRC302_clamp_max
SOC_soll = SOC_RL_Begrenzung = 10%



TIRC302_clamp_ = 50.7 TIRC302_clamp_max = 55
Szenario_TWW = 9
SOC_soll = 10 SOC_RL_Begrenzung = 10
FLAG_9_TWW = TRUE

TIRC302_clamp_ = 50.7 TIRC302_clamp_zul... = 50
FLAG_9_TWW = TRUE

(3)Rücklauftemperatur noch aktiv, da
TIRC302_clamp_ > TIRC_clamp_zulaessig;
Temperatur wird weiter verringert;
Ventil MV301 komplett geschlossen, da SOC_soll = 10%.



TIRC302_clamp_ = 49.8 TIRC302_clamp_max = 55
Szenario_TWW = 10
SOC_soll = 80 SOC_RL_Begrenzung = 10
FLAG_9_TWW = FALSE

TIRC302_clamp_ = 49.8 TIRC302_clamp_zul... = 50
FLAG_9_TWW = FALSE

(4)Normalbetrieb: Rücklauftemperaturbegrenzung deaktiviert, da Temperatur so weit verringert, bis
TIRC302_clamp_ < TIRC302_clamp_zulaessig
SOC_soll = SOC_Leitwarte = 80%
Ventil ca. ¼ geöffnet

Beschreibung der Validierung in der Hardwaretestumgebung anhand des Szenario *Rücklauftemperaturbegrenzung.* Die Fotos zeigen die simulierte Ventilstellung des Aktors MV301 (1,3,4) sowie das Verstellen des Widerstands TIRC302 (2).
Die Online-Anzeige der Werte in der Entwicklungsumgebung CoDeSys 2.3 ist in den gelben Kästen dargestellt.

Abbildung A.19.: Bilderstrecke zur Validierung in der Hardwaretestumgebung

Zusätzliche Erklärung der Validierung des Regelalgorithmus

Die folgenden Abschnitte enthalten zusätzliche Erläuterungen zu verschiedenen Problemstellungen bei der Validierung der einzelnen Szenarien des Regelalgorithmus. Die in den Grafiken enthaltenen Zeitangaben (t_1, t_2, ...) entsprechen denen der einzelnen Abschnitte aus Kapitel 6.6.2.

Leistungsbegrenzung (iWÜST mit TWW)

Die abgenommene Leistung der Anlage wird im Abstand von 30 Sekunden vom simulierten Wärmemengenzähler berechnet (siehe Kapitel 4.3.2). Der berechnete Wert wird anschließend in eine Liste des Formats *ARRAY* gespeichert. Das ARRAY besitzt 30 Einträge, die nach der Logik *Last-In-First-Out* (*LIFO*) beschrieben werden. Dadurch enthält das ARRAY immer die berechneten Leistungswerte der letzten 15 Minuten. Der Mittelwert aller Einträge entspricht dem Leistungsmittelwert der letzten 15 Minuten. Dadurch ergibt sich das stufenförmige Profil der Leistungsabnahme in Abbildung 6.5. Die vertraglich vereinbarte maximale Leistungsabnahme wurde mit 105 kW angenommen. Das Auslösen des Szenarios in der Simulationsumgebung *Jarvis* wurde unter Verwendung des *Testszenarios* provoziert, indem die maximale Leistungsabnahme der Last *load1* angepasst wurde. Ab dem Zeitpunkt t_1 beträgt die bezogene Leistung von *load1* 70 kW (siehe Abbildung A.20). Dadurch ergibt sich eine Gesamtlast der Anlage ab diesem Zeitpunkt von 135,5 kW. Dieser Wert liegt deutlich über dem vertraglich festgelegtem Wert mit Kulanz von 115,5 kW.

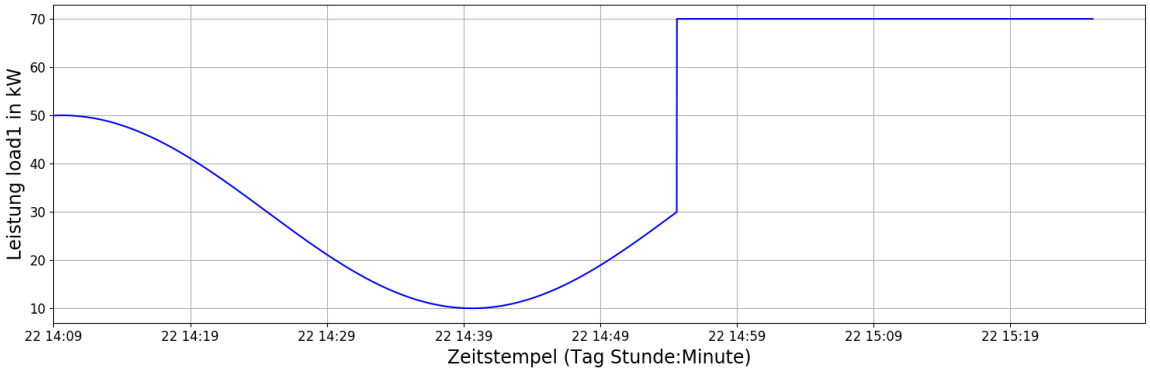


Abbildung A.20.: Lastkurve *load1* (Leistungsbegrenzung iWÜST mit TWW)

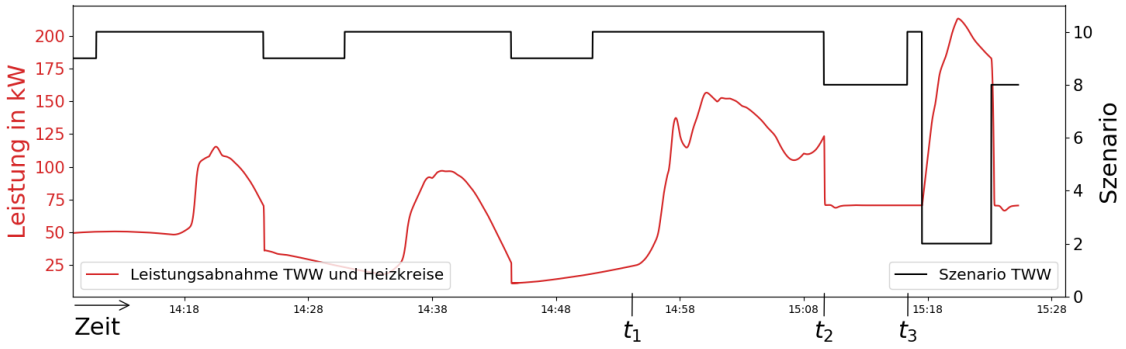


Abbildung A.21.: Leistungsabnahme (Leistungsbegrenzung iWÜST mit TWW)

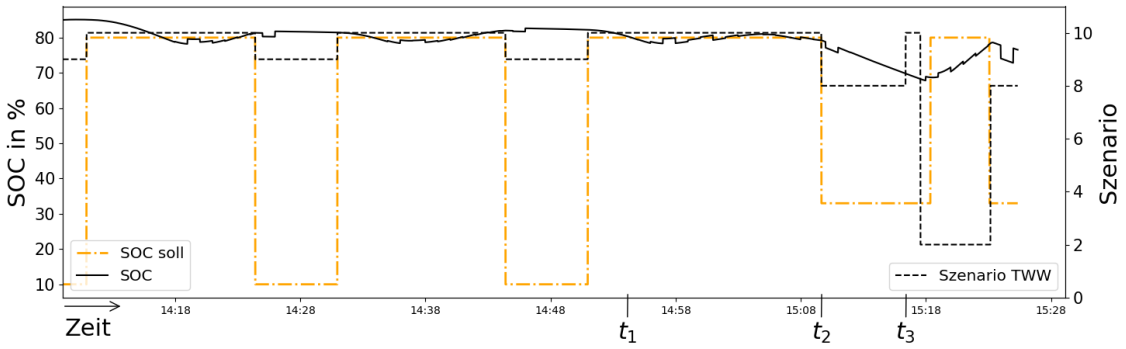


Abbildung A.22.: SOC (Leistungsbegrenzung iWÜST mit TWW)

Füllstandsbegrenzung (iWÜST mit TWW)

Eine minimale Öffnung des Ventils *MV301* sorgt dafür, dass ein minimaler Durchfluss am Temperatursensor *TIRC305* vorhanden ist und dieser den aktuellen Systemzustand im Rücklauf erfassen kann. In der Simulationsumgebung wurde das Szenario *Füllstandsbegrenzung* ausgelöst, indem zwei Parameter der *Testsimulation* angepasst wurden. Die minimale Leistungsabnahme von *load2* wurde von 7,3 kW auf 8 kW erhöht. Dadurch wird die maximale Rücklauftemperatur am Temperatursensor *TIRC302* begrenzt, sodass Szenario 9 nicht aktiviert wird. Diese Änderung vereinfacht die Interpretation der Simulationsergebnisse. Der Sollwert des SOC wurde für dieses Szenario auf 90 % festgelegt.
Die Ergebnisse zeigen, dass die Regelung der Pumpe *MV301* auf die Änderung des Werts *SOC_soll* um 40 % reagiert, indem der von der Pumpe *MV301* geförderte aktuelle Massenstrom von 1,7 kg/s innerhalb von sieben Zeitschritten auf den minimalen Massenstrom von 0,011 kg/s sinkt. Das hat zur Folge, dass der Massenstrom der Komponente *generator* innerhalb von 7 Zeitschritten um 89,5 % sinkt. Durch diese sprunghafte Änderung des Mas-

senstroms im System steigt die Temperatur am Temperatursensor *temp_sensor_help* auf über 80 °C nach Formel 2.3 an. Die Regelung des Wärmeerzeugers *pid_generator* reagiert auf diese Veränderung mit einem hochfrequenten Schwingen der Stellgröße. In Abbildung A.23 ist der Temperaturverlauf nach Aktivierung der *Füllstandsbegrenzung* für den Zeitbereich 14:42 Uhr bis 14:45 Uhr dargestellt. Auf die Regelung der Regelgrößen (*TIRC401*) in

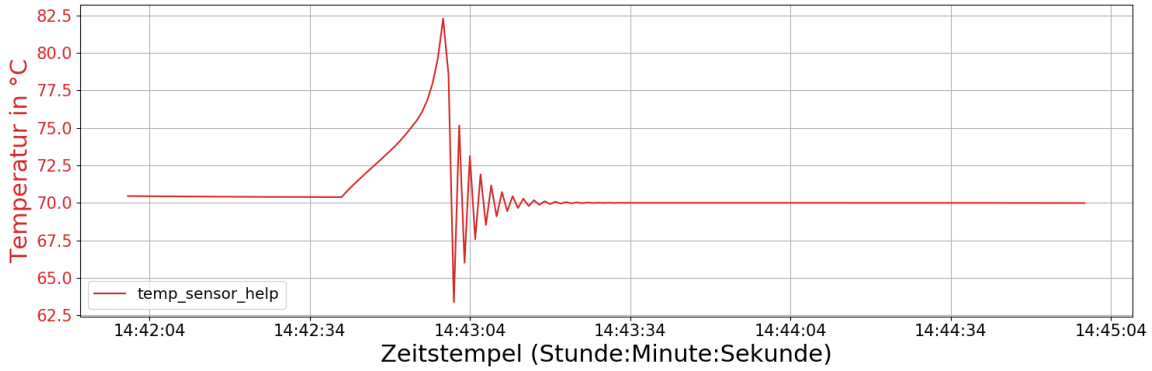


Abbildung A.23.: *temp_sensor_help* im Zeitraum 14:42 Uhr - 14:45 Uhr (Füllstandsbegrenzung)

der Simulationsumgebung hat dies nur geringe Auswirkungen, da die Temperatur weiterhin im Mittel dem Sollwert der Netztemperatur entspricht und die Schwingungen nicht auf die Wasserkreiseläufe *TWW* und *Heizkreise* übetragen werden.
In der Realität kann dieser Fall aufgrund der Trägheit des Systems und der Ventilstellzeiten nicht eintreten. In der Regel brauchen die Ventile für das Abfahren des gesamten Stellbereichs zwischen 60 s und 180 s. Dies ist bei der Verwendung der Pumpen im Modell nicht gegeben. Aufgrund des gering eingeschätzten Einflusses wird die Regelung nicht verändert.
Zur Validierung des Regelalgorithmus wurde dieser in der Entwicklungsumgebung *CoDeSys 2.3* angepasst. Das Einschwingen der Temperatur des Sensors *temp_sensor_help* (Abbildung A.23) hätte zur Folge, dass die Temperaturdifferenz *T_delta_Fuellstandsbegrenzung_out* direkt nach Eintreten in das Szenario überschritten wird und das Austrittskriterium erfüllt wäre. Deswegen wurde die Zählervariable *Zaehler_Fuellstandsbegrenzung* eingeführt, die bezweckt, dass das Austrittskriterium erst nach 20 s abgefragt wird, da sich die Temperatur am Sensor *temp_sensor_help* danach eingeschwungen hat (siehe Abbildung A.23). Dadurch ist sichergestellt, dass die Temperatursprünge im Netzvorlauf das Ergebnis nicht verfälschen.
Für die graphische Auswertung der Simulationsergebnisse wurde für die Netzvorlauftemperatur der konstante Wert 70 °C verwendet. Die Ausschläge der Temperatur am Sensor *temp_sensor_help* werden dadurch in der Graphik nicht dargestellt. Für die Validierung des Regelalgorithmus ist dies hinnehmbar, da sie im Programmcode durch die Zählvariable *Zaehler_Fuellstandsbegrenzung* ebenfalls keine Auswirkungen auf die Regelung haben. Der zeitliche Verlauf der Temperaturdifferenz und des Szenarios sind in Abbildung A.24 visuali-

siert.

Es ist zu erkennen, dass das Szenario bei Unterschreiten einer Temperaturdifferenz von

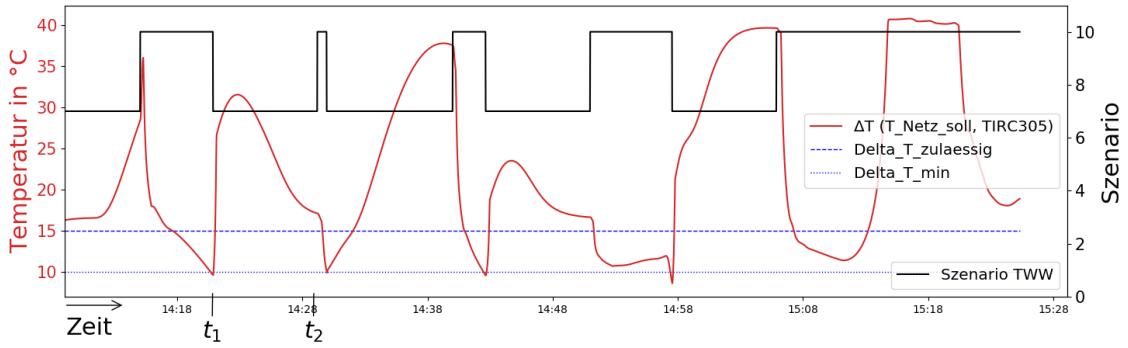


Abbildung A.24.: Temperaturdifferenz (T_Netz_soll, TIRC305; Füllstandsbegrenzung)

10 °C aktiviert wird (bspw. Zeitpunkt t_1). Anschließend steigt die Temperaturdifferenz stark an. Die Temperatur beträgt nach Aktivierung des Szenarios jene Temperatur, die vom Rohr *pipe11* in das T-Stück *t_piece6* übergeben wird. Durch die Rohrleitung und die Verringerung des Massenstoms ist eigentlich zu erwarten, dass die Temperatur vorerst konstant bleibt, bis der kältere Massenstrom aus dem Rücklauf des Wärmeübertragers durch das Rohr geschoben wurde. Bei der genaueren Untersuchung dieses Phänomens wurde ein Fehler im Modell des Elements *Pipe* erkannt. Die Eingangstemperatur wird in dieser Situation fälschlicherweise als Ausgangstemperatur ausgegeben. Dadurch entsteht der sprunghafte Temperaturanstieg am Sensor *TIRC305*. Es wird nach dem Verringern des Massenstroms die Eingangstemperatur der Komponente *pipe8* am Sensor *TIRC305* gemessen. Zur Validierung des Szenarios wurde deswegen die Komponente *pipe8* aus dem Modell entfernt. In den folgenden Kapiteln wird bei der Verwendung des *Testsimulations* immer auf die Komponente *pipe8* verzichtet. Dies hat keine Auswirkungen auf die im Folgenden beschriebenen Ergebnisse. Bei anderen Komponenten vom Typ *pipe* ist dieser Fehler nicht aufgetreten, weshalb davon auszugehen ist, dass der Fehler nur unter bestimmten Umständen auftritt, wobei dies nicht weiter untersucht wurde. Die folgende Validierung der Ergebnisse hat gezeigt, dass der Temperatursensor *TIRC305* grundsätzlich nicht für die Deaktivierung des Szenarios geeignet ist, weshalb ein Verzicht auf die Komponente zur leichteren Verständlichkeit der Ergebnisse vertretbar ist.

In Abbildung A.25 ist der zeitliche Verlauf des SOC bei Verwendung des angepassten Modells dargestellt. Dieser verändert sich nach Eintritt in das Szenario nur marginal, da in diesem Zeitraum nur eine geringe Lastabnahme vorhanden ist. Der von der Pumpe *MV301* geförderte Massenstrom ist in Abbildung A.26 gezeigt.

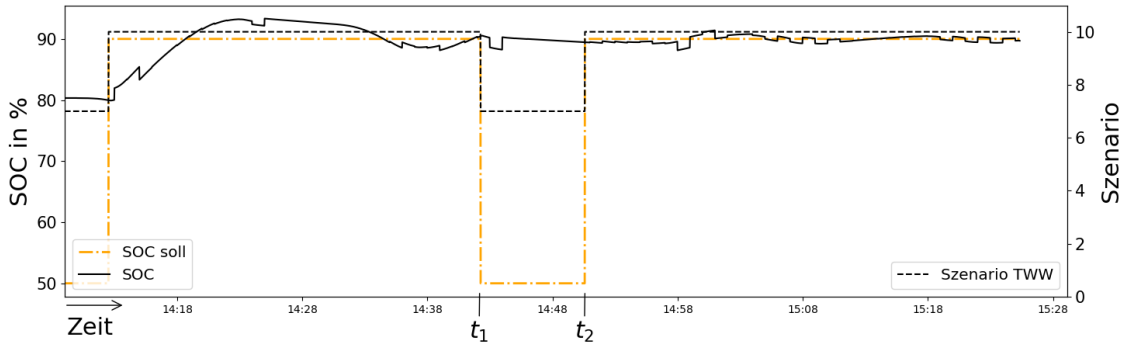


Abbildung A.25.: SOC ohne pipe8 (Füllstandsbegrenzung)

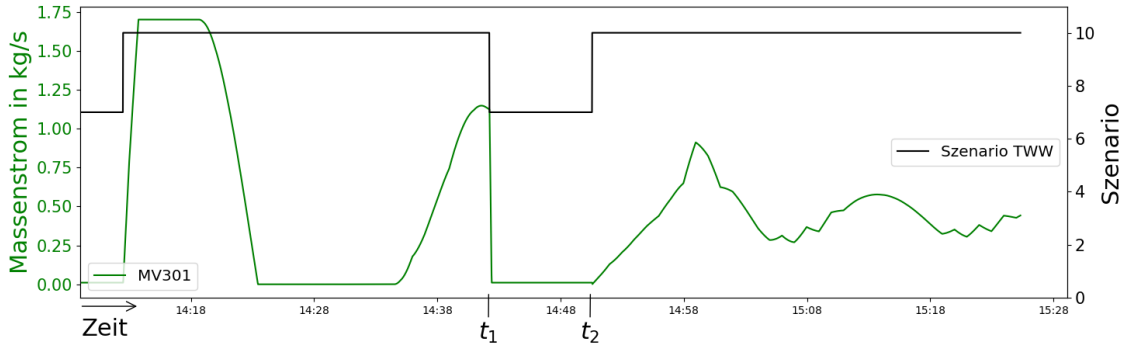


Abbildung A.26.: MV301 ohne pipe8 (Füllstandsbegrenzung)

Speicher Überladen (iWÜST mit TWW)

In der Simulationsumgebung ist kein Element verfügbar, das eine Leitwarte abbildet. Deshalb wurde das Setzen der Variable mit Hilfe einer Zählvariable im Anwenderprogramm umgesetzt. Zum Zeitpunkt t_1 wurde die Variable *EXTERN_Fuellstand_max_aktiv* von *FALSE* auf *TRUE* gesetzt. Zum Zeitpunkt t_4 wird die Variable auf *FALSE* zurückgesetzt. Die Auswirkungen auf den SOC sind in Abbildung 6.7 zu sehen. Durch das Auslösen des Szenarios steigt der SOC auf 100 %. Zum Zeitpunkt t_3 fällt die Temperatur am Sensor *temp_sensor_help* für ca. eine Minute auf 69,2 °C. Da die Berechnung des SOC abhängig von dem Mittelwert dieser Temperatur ist liegt der SOC in diesem Zeitbereich über 100 %.
Die Reaktion der Pumpe *MV301* ist anhand Abbildung A.27 erkennbar. Zum Zeitpunkt t_3 sinkt die Pumpenleistung wegen des SOC-Werts über 100 % kurzzeitig ab.

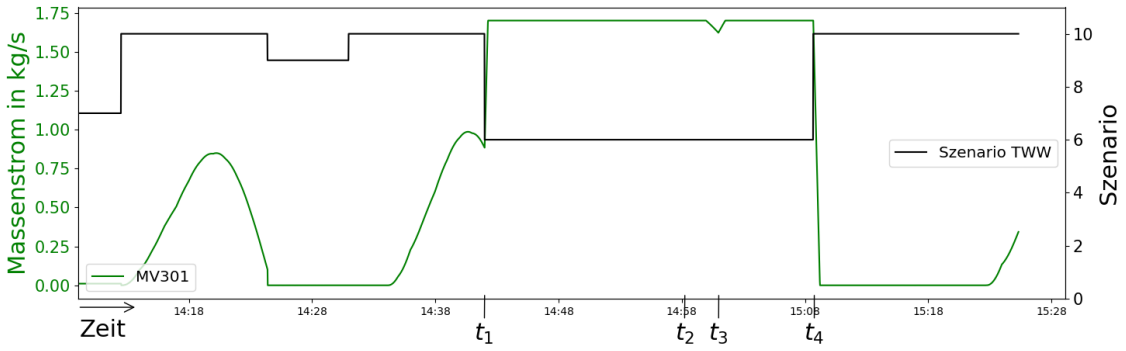


Abbildung A.27.: MV301 (Speicher Überladen)

Variante 2 des Netzkollaps (iWÜST mit TWW)

In der Regelbeschreibung wird vorgegeben, dass das Szenario 2 eine höhere Priorität als die Szenarien 3 und 4 besitzen. Da es sich jedoch bei den Szenarien 3 und 4 um Störfälle handelt und die übrigen Szenarien 5-10 eine niedrigere Priorität besitzen, muss für andere Situationen keine Speicherkapazität zurückgehalten werden. Aus diesem Grund wurde der Regelalgorithmus für eine weitere Simulation bzgl. der Prioritäten so verändert, dass die Szenarien 3 und 4 höher priorisiert sind als Szenario 2. Dadurch sollte erreicht werden, dass die Speicherkapazität besser ausgenutzt wird. Der Regelalgorithmus wurde also dahingehend verändert, dass der Speicher bei Szenario 3 oder 4 so lange geleert wird, bis der Sollwert des SOC (10 %) erreicht ist oder die Temperaturdifferenz zwischen oberster Speichertemperatur und Sollwert der Temperatur am Sensor *TIRC401* kleiner als 3 °C ist. Sobald diese Temperaturdifferenz unterschritten wird, wird dem Sollwert des SOC (*SOC_Soll*) der zu diesem Zeitpunkt berechnete SOC übergeben. Dadurch soll sichergestellt werden, dass der Speicher weder geleert noch befüllt wird und der Wärmeübertrager nicht mit einem Wärmeträgermedium versorgt wird, welches kälter als der Sollwert des Reglers *pid_P401* ist. Die angepasste Regelstrategie wurde anschließend unter Verwendung des Testsimulations erprobt. Die zeitliche Darstellung des SOC in Abbildung A.28 zeigt, dass das gewünschte Verhalten nach Setzen der Variable *EXTERN_Netzkollaps* zum Zeitpunkt t_1 vorerst eintritt. Zum Zeitpunkt t_3 sinkt die Temperatur am Sensor *TIRC324* unter den Grenzwert *TIRC401_soll* +3 °C (siehe Abbildung A.29). Ab diesem Zeitpunkt beträgt der Sollwert des SOC rund 67 %. Dieser Wert entspricht dem aktuell berechneten SOC. Zum Zeitpunkt t_4 wird die Variable *EXERN_Netzkollaps* auf den Wert *FALSE* zurückgesetzt. Das Szenario 2 wird, im Gegensatz zur vorherigen Regelstrategie, nicht aktiviert, solange Szenario 4 aktiviert ist. Nach Deaktivierung von Szenario 4 wird sofort Szenario 2 aufgrund der geringen oberen Speichertemperaturen aktiviert. In Abbildung A.29 ist der zeitliche Verlauf der Temperatur des Sensors *TIRC401* und des Sensors *TIR301* gezeigt. Anhand der Graphik

wird deutlich, dass die im Speicher vorgehaltene Menge an Wärmemenge nicht ausreichend ist, um die primäre Sollvorlauftemperatur am Sensor *TIRC401* zu erreichen. Abbildung A.31 verdeutlicht dies anhand des Massenstroms von Pumpe *P401* und dem Temperaturverlauf des Sensors *TIRC401*. Trotz der maximal möglichen Fördermenge von Pumpe *P401* wird die Solltemperatur bei der Leistungsabnahme nicht erreicht.

Das Problem dieser Regelstrategie ist, dass der Sollwert des SOC nicht perfekt von der

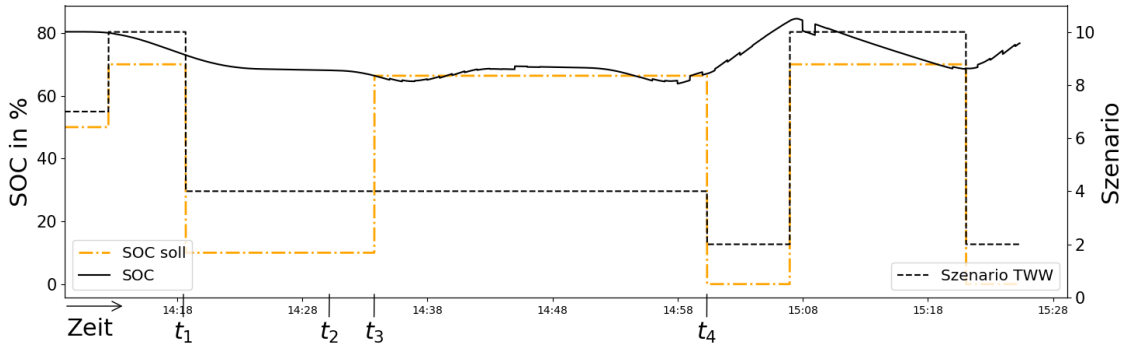


Abbildung A.28.: SOC (Variante 2; Netzkollaps; iWÜST mit TWW)

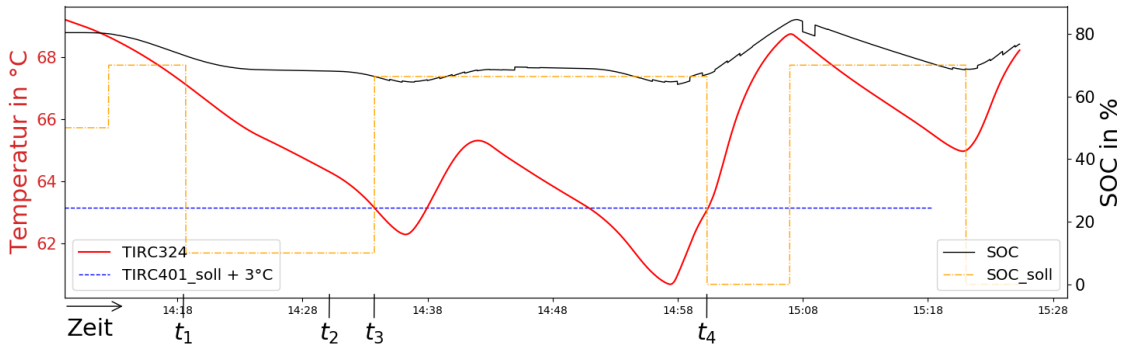


Abbildung A.29.: SOC, *TIRC324* (Variante 2; Netzkollaps; iWÜST mit TWW)

Pumpe *MV301* gehalten werden kann. Durch die Schwankungen wird der Speicher zeitweise also trotzdem entladen. Dadurch können zu niedrige Temperaturen in den primärseitigen Vorlauf der Wärmeübergabestation gelagen, wodurch die Temperatur am Sensor *P401* nicht mehr gehalten werden kann.

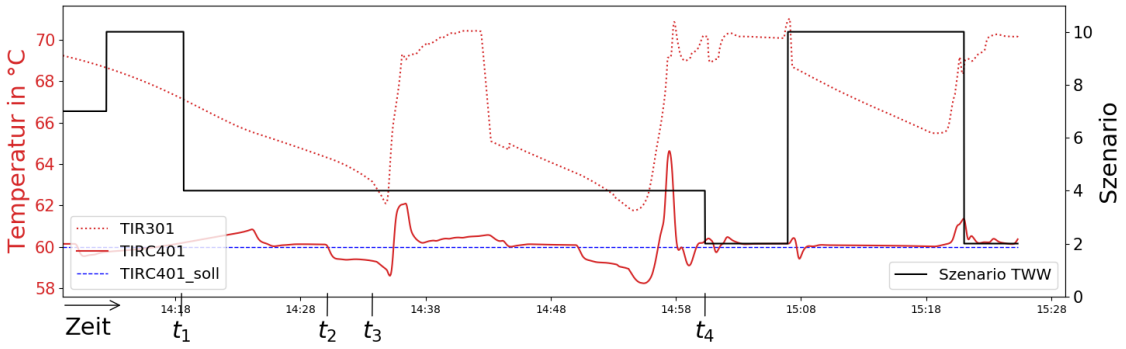


Abbildung A.30.: *TIR301*, *TIRC401* (Variante 2; Netzkollaps; iWÜST mit TWW)

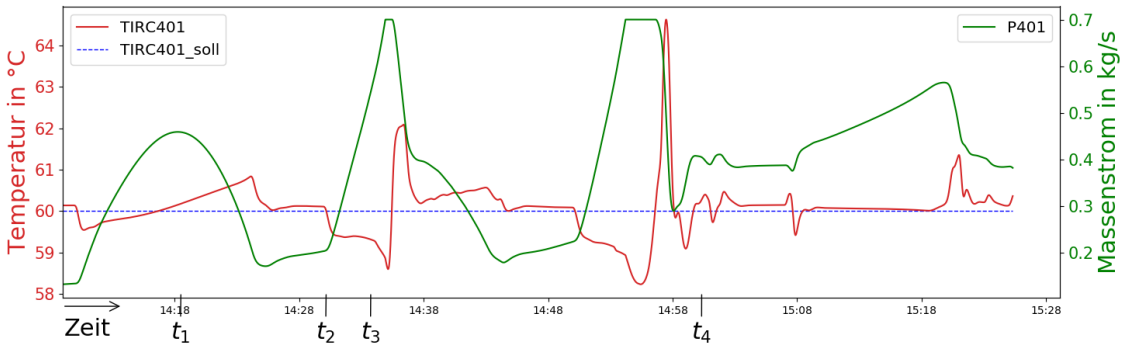


Abbildung A.31.: *TIRC401*, *P401* (Variante 2; Netzkollaps; iWÜST mit TWW)

Minimale Temperatur (iWÜST mit TWW)

Die folgenden Simulationsergebnisse wurden bei einer Netzemperatur von 70 °C generiert. Der Sollwert *SOC_soll* wurde dafür im Zeitraum t_1 bis t_5 auf 60 % verringert. Das vorgesehene Verhalten bzgl. der Grenzwerte zum Ein- und Austritt des Szenarios ist in Abbildung A.32 dargestellt.
Die Auswirkungen auf den Soll- und Istwert des SOC sind in Abbildung A.34 dargestellt. Der SOC steigt, sobald Szenario 2 ausgelöst wurde. Die Veränderung der Regelgröße wird durch den Sollwert *SOC_soll* = 0 % dargestellt. Der zeitliche Verlauf des SOC, SOC_soll und der Szenarien für die Netztemperatur 80 °C ist in Abbildung A.34 dargestellt.

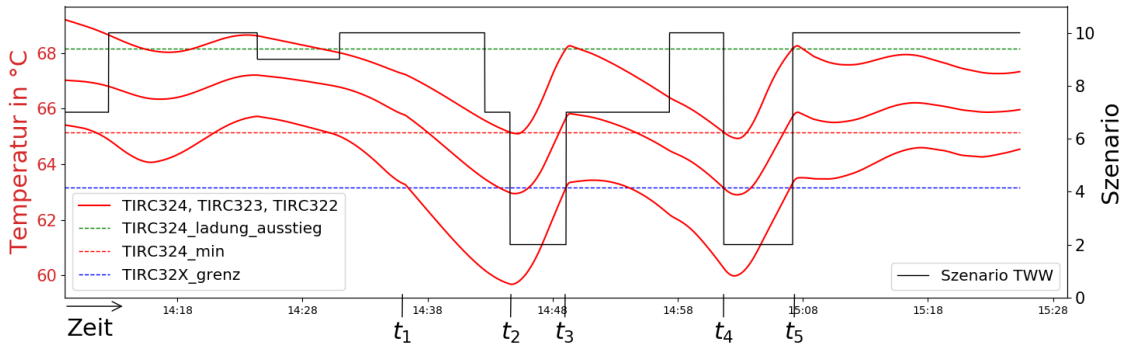


Abbildung A.32.: *TIRC324*, *TIRC323*, *TIRC322* (Minimale Temperatur; iWÜST mit TWW, T_Netz=70 °C)

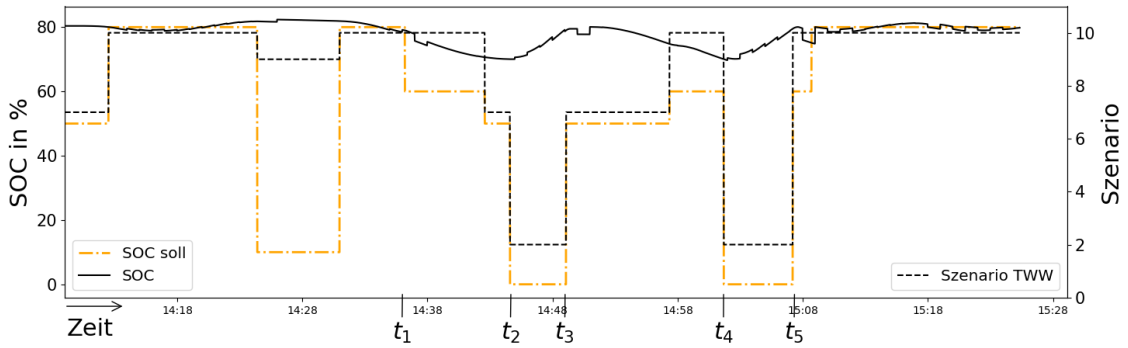


Abbildung A.33.: SOC (Minimale Temperatur; iWÜST mit TWW, T_Netz=70 °C)

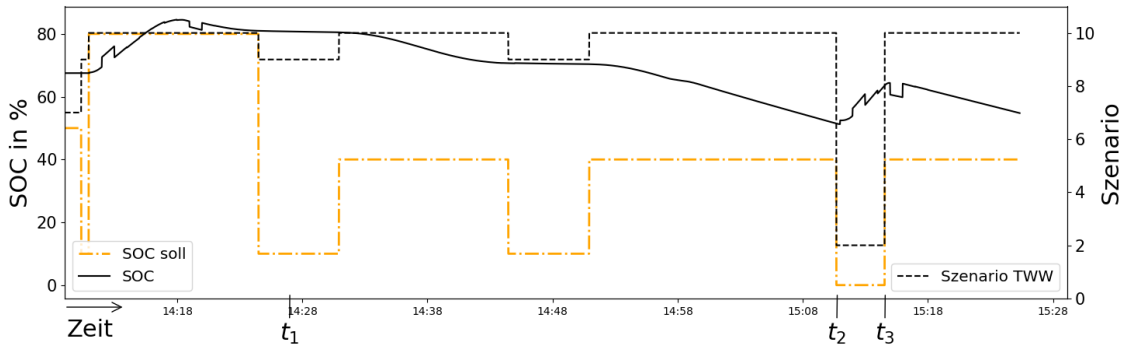


Abbildung A.34.: SOC (Minimale Temperatur; iWÜST mit TWW, T_Netz=80 °C)

Programmspezifische Zuweisungen der iWÜST mit TWW

Im Code des Programms *PLC_PRG_TWW()* folgt auf die einzelnen Codeteile zu den Szenarien die Zuweisung der Soll- und Istwerte für den PID-Regler *pid_MV301*. Dies ist im Programmablaufplan in Abbildung A.35 dargestellt. Ist Szenario 2 aktiv (und somit die Variable *FLAG_2_TWW* = TRUE), so wird der Stellgröße des PID-Reglers *pid_MV301* die Temperatur *TIRC324* zugewiesen. Ansonsten ist die Stellgröße des Reglers der SOC. Am Ende des Codes werden die Datentypen der Soll- und Istwerte für die Verwendung im Baustein *PID* angepasst.

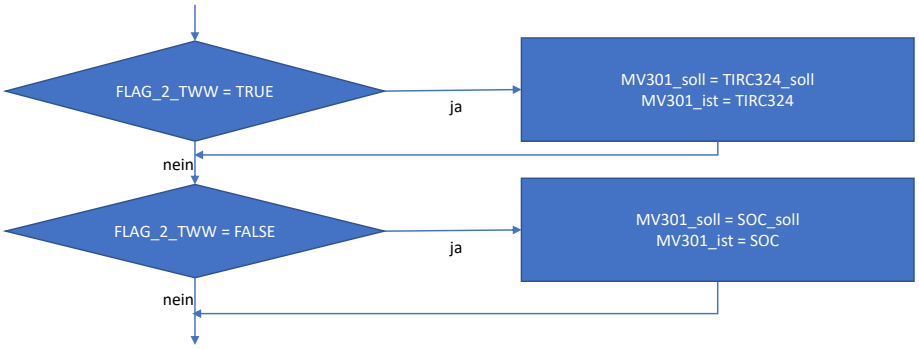


Abbildung A.35.: Programmablaufplan der Zusweisungen des PID-Reglers *pid_MV301*

Normalbetrieb (iWÜST ohne TWW, Sollwert TC101)

Das gewünschte Verhalten des Sollwerts *T101_soll* kann anhand der Simulationsergebnisse in Abbildung A.36 gezeigt werden. Die Parameter für die Bestimmung von *TC101_soll* wurden während der Simulation in den definierten Zeiträumen wie in Tabelle A.1 dargestellt verändert. Die Sollwertveränderungen im Zeitraum t_1 bis t_2 werden durch das höher priorisierte Szenario *Rücklaufbegrenzung* verursacht und stehen in keinem Zusammenhang mit den Werten aus Tabelle A.1. Der Sollwert *TC101_soll* im Zeitraum t_4 bis t_5 wird anhand der Formel *TC101_soll* = *T_Netz_ist* - *WÜT* berechnet.

Tabelle A.1.: Einstellungen der Variablen zur Überprüfung der Sollwertvergabe *TC101_soll* (Normalbetrieb)

Zeitraum	TC101_vertrag_soll	TC101_Kunde_soll	TC101_Leitwarte_soll
Start - t_1	67 °C	80 °C	70 °C
t_1 - t_2	67 °C	80 °C	63 °C
t_2 - t_3	72 °C	80 °C	63 °C
t_3 - Ende	72 °C	67 °C	63 °C

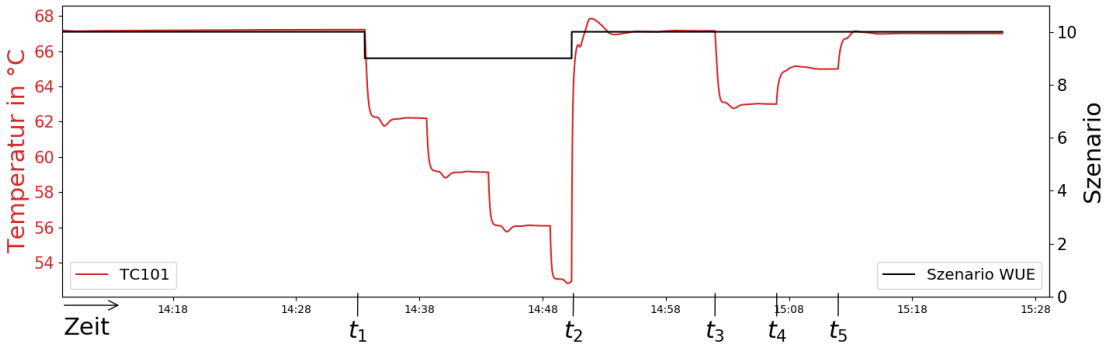


Abbildung A.36.: *TC101* (Normalbetrieb; iWÜST ohne TWW)

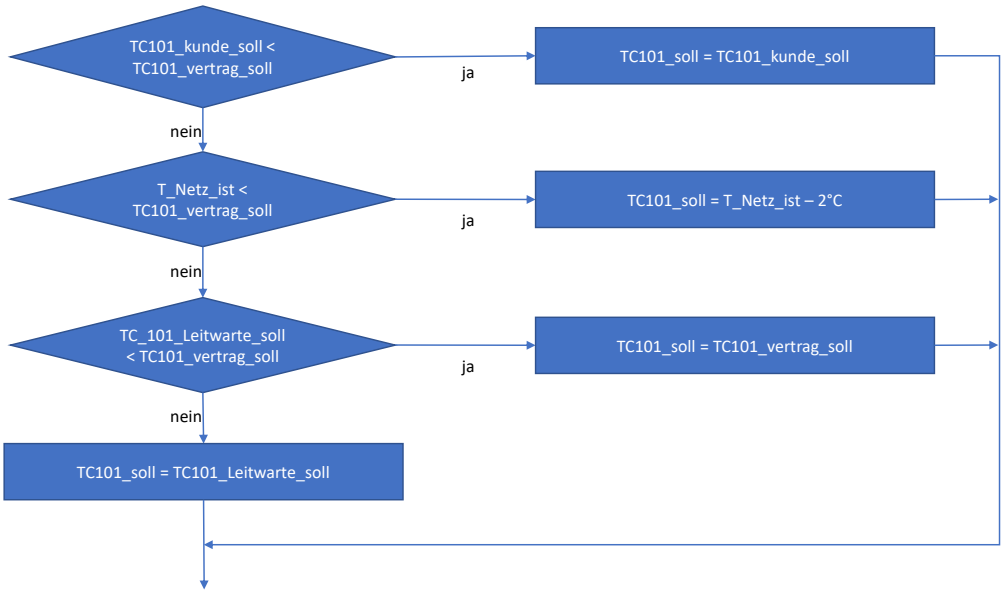


Abbildung A.37.: Programmablaufplan Sollwertvergabe *TC101_soll* (Normalbetrieb)

Rücklaufbegrenzung (iWÜST ohne TWW)

Variante 1: Kontinuierliche Rücklaufregelung auf einen Sollwert

Sobald Szenario 9 ausgelöst wird, verändern sich bei dieser Variante die Regelgröße sowie die Proportionalverstärkung des Reglers *pid_MV101*. Das Stellglied *MV101* wird nun verwendet, um die Temperatur des Sensors *TIRC103* (Istwert) zu verringern. Als Sollwert wird der Wert *TIRC103_zulaessig* $-2\,°C$ dem Regler übergeben. Dadurch regelt der Aktor *MV101* auf eine Stellgröße im gleichen Regelkreis wie der Regler *pid_generator*. Um zu vermeiden, dass ein Resonanzverhalten zwischen den Reglern auftreten kann, wurde dem Regler *pid_MV101* eine starke Dämpfung in Form eines kleinen Proportionalwerts $Kp = 2 \cdot 10^{-7}$ zugewiesen. Die Regelungsparameter wurden anschließend nicht weiter angepasst, da anhand dieser Parametrierung deutlich gezeigt werden kann, dass diese Regelungsvariante für die Anforderungen nicht geeignet ist.
Anhand den Abbildung A.38 wird die Variante im Folgenden analysiert. Zum Zeitpunkt t_1 wird das Szenario 9 ausgelöst, da die Temperatur am Sensor *TIRC103* den Grenzwert *TIRC103_max* übersteigt. Der Temperaturverlauf ändert sich im Anschluss nicht sofort, da die Rohrleitung *pipe3* im regelungstechnischen Sinn ein Totzeitglied darstellt. Die starke Dämpfung sorgt ebenfalls dafür, dass die Temperatur anschließend nur langsam sinkt. Szenario 9 wird deaktiviert, sobald die Temperatur unter den Wert *TIRC103_zulaessig* fällt (Zeitpunkt t_2). Die Temperatur steigt nach Ablauf der Totzeit wieder stark an, sodass der obere Grenzwert überschritten wird (Zeitpunkt t_3) und Szenario 9 wieder aktiviert wird. Wegen der Minimierung des primärseitigen Durchflusses am Wärmeübertrager fällt die Temperatur am Sensor *TC101* ab. Sobald Szenario 9 deaktiviert wurde, wird dem Regler *pid_MV101* wieder die Regelgröße *TC101* übergeben. Dies ist anhand des Temperaturverlaufs am Sensor *TC101* zum Zeitpunkt t_4 in Abbildung A.38 ersichtlich.
Die Auswertung macht deutlich, dass für die Begrenzung der Rücklauftemperatur diese

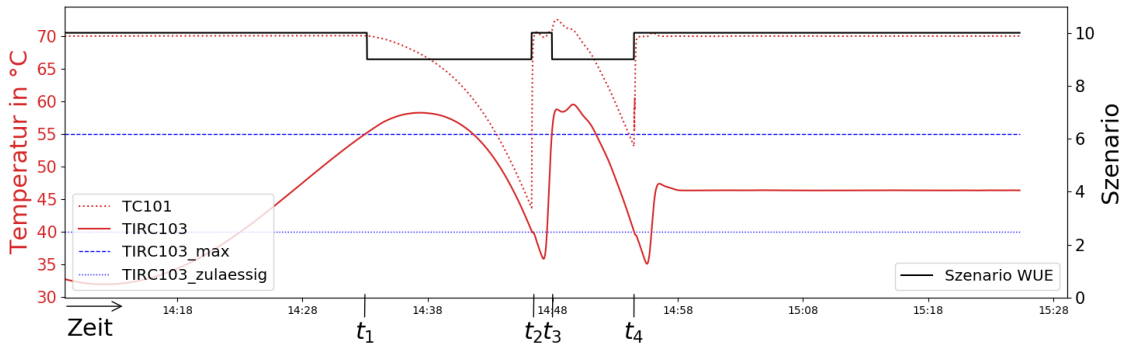


Abbildung A.38.: *TC101*, *TIRC103* (Variante 1; Rücklaufbegrenzung; iWÜST ohne TWW)

Variante nicht den gewüschten Effekt hat. Grundsätzlich hat die Regelungsstrategie zunächst eine Verringerung der Temperatur am Sensor *TIRC103* zur Folge. Jedoch verringert sich diese Temperatur nur so lange, bis die Temperatur unter den Grenzwert des Austrittskriterium (*TIRC103_zulaessig*) sinkt. Wenn die Leistungsabnahme sich währenddessen wie im *Testsimulation* nur geringfügig ändert, wird das Szenario anschließend wieder aktiviert, sodass ein Pendeln zwischen den Szenarien entstehen könnte. Bei einer Parametrierung der Regelung, die ein schnelleres Erreichen des Sollwerts bedingt, würde das Szenario 9 in kürzeren Abständen aktiviert und deaktiviert werden.

Variante 2: Minimaler Öffnungswinkel des Ventils

Sobald das Eintrittskriterium von Szenario 9 erfüllt ist, wird in dieser Variante der Massenstrom am Aktor *MV101* langsam auf den minimalen Wert vermindert. Aufgrund der Erkenntnisse aus den Simulationen mit Variante 1 wurde der Timer *Timer_RL_Begrenzung_WUE* integriert. Dieser verhindert, dass im Bereich der geringen Leistungsabnahme von *load1* das Szenario 9 mit einer hohen Frequenz aktiviert und deaktiviert wird. Die Lastkurve der Komponente *load1* wurde für diese Simulation insofern verändert, als dass die abgenommene Wärmeleistung ab dem Zeitpunkt t_3 3 kW beträgt. Dadurch kann die Reaktion der Regelung bei einem starken Lastabfall und einer dadurch bedingten hohen Temperatur am relevanten Sensor *TIRC103* verdeutlicht werden.
Abbildung A.39 stellt den zeitlichen Verlauf der relevanten Parameter für Szenario 9 dar. Es zeigt sich, dass die Temperatur am Sensor *TIRC103* durch den Regelalgorithmus mit Variante 2 nach Aktivierung des Szenarios (Zeitpunkt t_1) verringert wird. Der Timer *Timer_RL_Begrenzung_WUE* wird gestartet, sobald die Temperatur des Sensors *TIRC103* den Grenzwert *TIRC103_zulaessig* unterschritten hat (Zeitpunkt t_2). Zehn Minuten später wird Szenario 9 deaktiviert. Anschließend steigt die Temperatur sprunghaft an, da das Ausgangssignal des Reglers *pid_MV101* wegen der großen Regelabweichung zwischen Sollwert (*TC101_soll* = 70 °C) und Istwert (*TC101* = 20 °C) an. Zum Zeitpunkt t_3 übersteigt die Temperatur am Sensor *TIRC103* den Grenzwert abermals aufgrund des Lastabfalls von *load2*, sodass Szenario 9 wieder bis zum Ende der Simulation aktiviert ist.

Aus den Ergebnissen der beiden vorherigen Simulationen ist erkennbar, dass die Temperatur am Sensor *TIRC103* als Austrittskriterium für dieses Szenario nicht ausreicht. Die Temperatur im Rücklauf sinkt zunächst auf die Temperatur *TIRC103_zulaessig*. Nach Erfüllen des Austrittskriterium steigt sie jedoch erneut stark an, bis das Szenario wieder aktiviert ist, falls sich die Lastabnahme zwischenzeitlich nicht deutlich erhöht hat. Durch den Timer wird dieses Problem zwar entschärft, jedoch würde die Leistung bspw. bei einem kurzzeitigen Abfallen unter den Grenzwert für einige Sekunde trotzdem für 10 Minuten zurückgehalten werden,

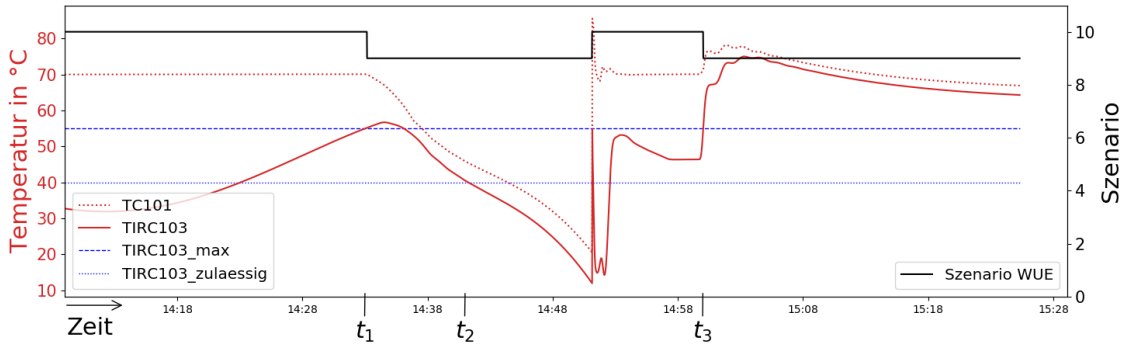


Abbildung A.39.: *TC101*, *TIRC103* (Variante 2; Rücklaufbegrenzung; iWÜST ohne TWW)

sodass dem Kunde für 10 Minuten nicht die Temperatur geliefert wird, welche ihm eigentlich zustände.

Variante 3: Regelung der Temperatur des Sensors TC101

Detaillierte Beschreibung der Regelung von Variante 3:
Das Szenario wird ausgelöst, sobald die Temperatur am Sensor *TIRC103* den Grenzwert *TIRC103_max* überschreitet. Als Austrittskriterium wird neben der Rücklauftemperatur am Sensor *TIRC103* die Temperaturdifferenz der Sensoren *TIRC102* und *TC101* herangezogen. Sobald die Temperaturdifferenz zwischen sekundärseitigem Vor-und Rücklauf größer als *dT_RLbegrenzung_grenz* ist und die primärseitige Rücklauftemperatur *TIRC103* des Wärmeübertragers den Grenzwert *TIRC103_zulaessig* unterschreitet, wird das Szenario deaktiviert. Die Solltemperatur *TC101_soll* wird bei Auslösen des Szenarios um 5 °C verringert. Im Anschluss wird die Solltemperatur *TC101_soll* im Abstand von 5 Minuten so lange um 3 °C verringert, bis das Austrittskriterium erfüllt ist.

Leistungsbegrenzung (iWÜST ohne TWW)

Beschreibung der Problematik bzgl. der Leistungsabnahme und des Modells:
Leistung *load1*: 65 kW über den gesamten Simulationszeitraum; Leistung *load2*: 55 kW von t_2 bis Simulationsende
In Abbildung A.40 ist ersichtlich, dass das Szenario bei Überschreiten des Grenzwerts der maximalen Leistungsbegrenzung inklusive der Kulanz zum Zeitpunkt t_3 ausgelöst wird. Der Mittelwert der Leistung steigt bis zum Zeitpunkt t_1 an, da zuvor Leistungswerte, die zur

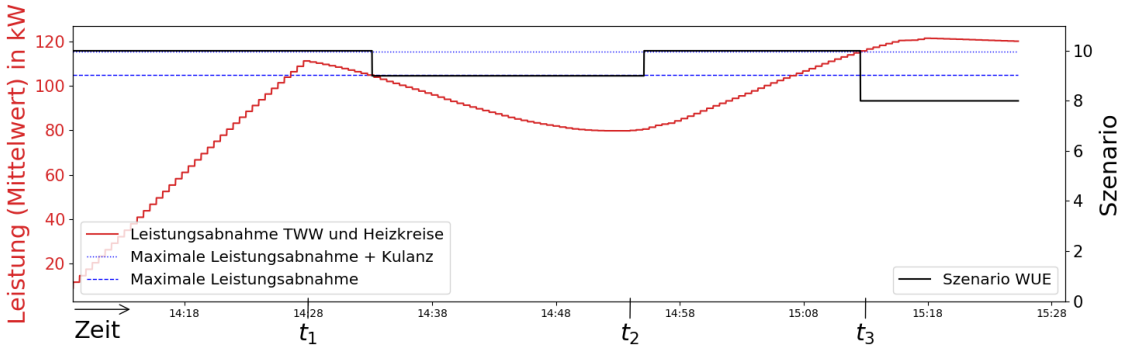


Abbildung A.40.: Mittelwert der Leistung (Leistungsbegrenzung; iWÜST ohne TWW)

Bestimmung des Mittelwerts verwendet werden, noch den Initalwert 0 betragen. Anschließend folgt der Mittelwert der Lastkurve von *load1*. Im Zeitraum t_2 bis t_3 steigt der Mittelwert, anschließend beträgt er konstant 120 kW.
Die Leistungsabnahme wird durch das Auslösen des Szenarios nicht verringert.Dieses Verhalten ist in Abbildung A.41 anhand der Temperatur *TIRC103* und des Massenstroms *MV101* ab dem Zeitpunkt t_3 bis zum Simulationsende deutlich erkennbar. Der Massenstrom der Pumpe *MV101* und die Temperatur am Sensor *TIRC103* sinken beide ab, weshalb sich die vom Wärmemengenzähler erfasste Leistung, die direkt proportional zum Massenstrom und der Temperaturdifferenz ist, nicht verändert. In einem realen System kann die Austrittstemperatur nicht unter die Umgebungstemperatur fallen.

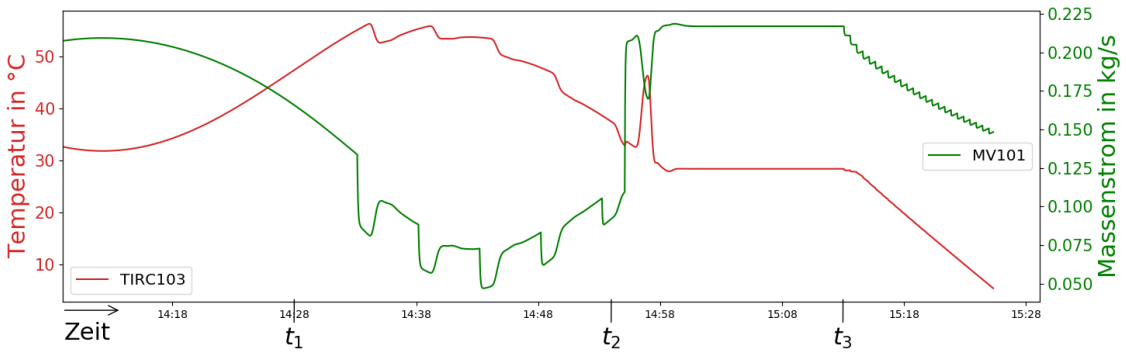


Abbildung A.41.: *MV101*, *TIRC103* (Leistungsbegrenzung; iWÜST ohne TWW)

Rücklauftemperaturregelung (iWÜST ohne TWW)

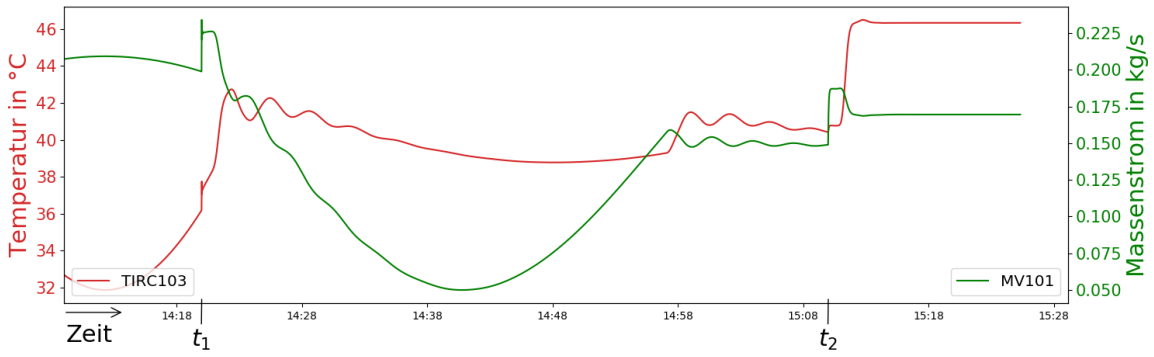


Abbildung A.42.: *MV101*, *TIRC103* (Rücklauftemperaturbegrenzung; iWÜST ohne TWW)

Netzkollaps (iWÜST ohne TWW)

Das Szenario *Netzkollaps* wird durch das Setzen der Variable *EXTERN_Netzkollaps* von der Leitwarte auf *TRUE* aktiviert und durch Setzen auf *FALSE* deaktiviert. Die Regelbeschreibung sieht vor, dass der Sollwert *TC101_soll* nach der Aktivierung des Szenarios nicht mehr außentemperaturabhängig ist. Der Sollwert wird entweder von der Leitwarte vorgegeben (*TC101_Netzkollaps_soll*) oder abhängig von der Netztemperatur und einer vorgegebenen Variable für die Grädigkeit (*T_WUT_graedigkeit*) berechnet. Der daraus resultierende Wert wird anschließend in der Abfrage bzgl. der Temperaturprioritäten aus Abbildung **??** für die Variable *T_Netz* eingesetzt. Anschließend wird der daraus ermittelte Wert als Sollwert *TC101_soll* an den Regler *pid_MV101* übergeben und von diesem eingestellt.

Warmhaltung (iWÜST ohne TWW)

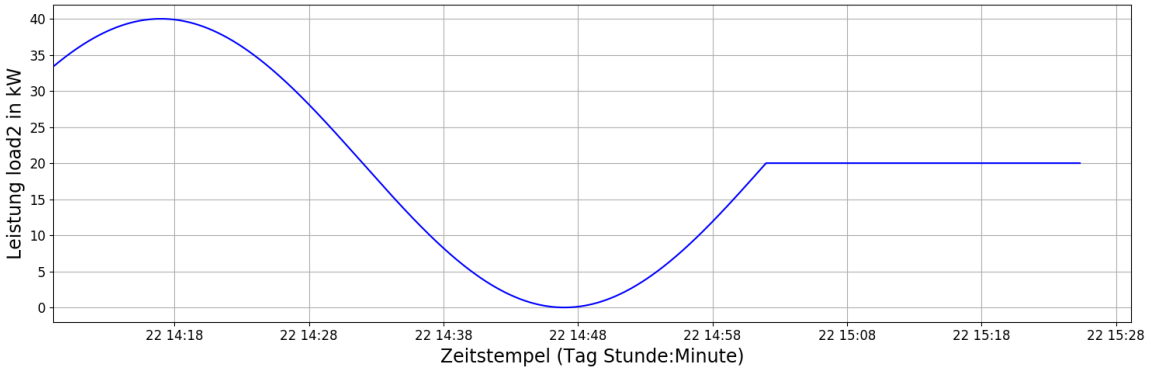


Abbildung A.43.: Leistungsabnahme *load2* (Warmhaltung)

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 6. Mai 2019	
Ort, Datum	Unterschrift