technische universität dortmund

IT in Produktion und Logistik

Entwicklung einer Monitoringumgebung zur Sorptionsberechnung von Polyamid 6 Probekörpern

Masterarbeit zur Erlangung des Grades M. Sc.

Vorgelegt von: Enno Henn

Matrikelnummer: 168817
Studiengang: Maschinenbau

ausgegeben am: 17.06.2022
eingereicht am: 31.10.2022

Erstprüfer: Dr.-Ing- Dipl. Inform. Anne Antonia Scheidler
Zweitprüfer: Anna Katharina Sambale, M.Sc.

Technische Universität Dortmund
Fakultät Maschinenbau
Fachgebiet IT in Produktion und Logistik

Leibniz-Institut für Polymerforschung Dresden e.V.
Institut Polymerwerkstoffe
Abteilung Werkstofftechnik
www.ipfdd.de

Masterarbeit

Entwicklung einer Monitoring-Umgebung zur Sorptionsberechnung von Polyamid 6 Probekörpern

Polyamid 6 (PA 6) wird in einer Vielzahl von anspruchsvollen technischen Anwendungen eingesetzt, insbesondere jedoch bei erhöhten thermischen sowie mechanischen Belastungen und im direkten Kontakt mit unterschiedlichen Medien. In der Anwendung erfährt PA 6 dabei eine stetige Veränderung der umgebenden Luftfeuchtigkeit, welche es absorbiert bzw. bei geringerer Luftfeuchtigkeit auch wieder abgibt. Die Diffusionsvorgänge des Wasser aufgrund der Hygroskopie des Materials werden durch verschiedene Parameter wie u.a. Temperatur, Umgebungsfeuchtigkeit und Belastung stark beeinflusst. Darüber hinaus ändern sich die mechanischen Kennwerte des Materials wie die Steifigkeit und die Zähigkeit durch die Wasseraufnahme bzw -abgabe stetig. Insbesondere die sich zeitlich und örtlich stetig verändernden Feuchtigkeitsverteilungen über den Probekörperquerschnitt sind prüftechnisch nur mit hohem Zeitaufwand zu charakterisieren, sodass eine numerische Methode zur Berechnung von Feuchtigkeitsverteilungen in PA-Bauteilen entwickelt und bereits validiert wurde. Diese Berechnungsmethode ist zum aktuellen Zeitpunkt in der Lage, mit einer zunächst zeitlich konstanten Umgebungsfeuchtigkeit eine Feuchtigkeitsverteilung innerhalb einer Probengeometrie während der Sorption zu berechnen.

Im Rahmen der Masterarbeit soll eine Probenmonitoring-Umgebung in Form von Raspberry Pi basierten Sensorstationen entwickelt werden, welche die umgebenden klimatischen Bedingungen von PA-Probekörpern ab der Probenherstellung über die verschiedenen Konditionierungsschritte sowie während der Lagerung bis hin zur Charakterisierung überwachen und die Daten an ein bestehendes Industrie 4.0 Softwaresystem (Symate Detact) weitergeben sollen. Jeder einzelne Probekörper soll hierbei eindeutig identifiziert werden. Die stationären Kennwerte wie bspw. Material und Angaben zur Probenherstellung etc. sollen direkt mit dem Symate Detact System verknüpft werden. Zudem soll für jeden Probekörper instationäre Größen wie die Umgebungsklimadaten kontinuierlich sowie aufgrund der Wasseraufnahme zu konkreten Messzeitpunkten innerhalb des Symate Detact Systems aufgenommen werden. Diese zeitabhängigen Feuchtedaten werden in Form von Randbedingung zur numerischen Berechnung der vorliegenden Feuchteverteilung mittels Massendiffusionsanalyse in ABAQUS berechnet. Eine Validierung der Berechnungsergebnisse soll mittels geeigneter Messmethode erfolgen.

Aufgabenstellung:

- Einarbeitung in den aktuellen Stand der Technik (Literaturrecherche)
- Aufbau einer Monitoring-Umgebung (Sensorstationen) zur Überwachung des Umgebungsklimas der Probekörper von der Herstellung über Temperschritte, Lagerung und Transport bis zur Charakterisierung
- Anschluss der entwickelten Monitoring Umgebung an ein bestehendes Industrie 4.0 Softwaresystem (Symate Detact)
- Einarbeitung in die Sorptionsberechnung auf Basis der Massendiffusionsanalyse mittels ABAQUS
- Verwendung der mittels Probenmonitoring erlangten zeitabhängigen Feuchtigkeitsdaten als Randbedingungen zur Sorptionsberechnung von Proben verschiedener Feuchtehistorien
- Validierung der Ergebnisse mittels geeigneter Messmethode
- Erstellen einer Nutzungsanleitung und Dokumentation der Auswertemethode
- Darstellung und Diskussion der Ergebnisse
- Anfertigung einer wissenschaftlichen Arbeit
- Präsentation der Ergebnisse

Ansprechpartner:
Anna-Katharina Sambale
Hohe Straße 6, Raum H206
01069 Dresden
Tel.: 0351/4658-1235
Email: sambale@ipfdd.de

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
AR	Argumented Reality
BUS	Binary Unit System
CAD	Computer Aided Desigen
CSI	Camera Serial Interface
CoAP	Constraind Application Protocol
DSI	Display Serial Interaface
FDM	Fused Deposition Modeling
FE	Finite Elemente
FEM	Finite Elemente Methode
GPIO	General Purpose Input / Output
GUI	Graphical User Interface
HMI	Human Machine Interface
HTTP	Hypertext Transfer Protocol
I2C	Inter-Intergrated Circuit
IIoT	Industrial Internet of Things
Iot	Internet of Things
IP	Internet Protocol

JSON	JavaScript Object Notation
KI	Künstliche Inteligenz
LAN	Local Area Network
M2M	Machine to Machine
MAC	Mandatory Access Control
MES	Manufacturing Execution System
MQTT	Message Queuing Telemetry Protocol
NTP	Network Time Protocol
OPC UA	Open Platform Communications United Architecture
PA	Polyamid
PA6	Polyamid 6
PC	Personal Computer
PLA	Polylactid
PLC	Programmable Logic Controler
QoS	Quality of Service
REST	Representational State Transfere
RFID	Radio Frequency Identification
r.H.	relativ Humidity
SCADA	Supervisory Control and Data Acquisition
SSH	Secure Shell
SSID	Service Set Identifier
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

UI	User Interface
UML	Unified Modeling Language
UUID	Universally Unique Identifier
V-LAN	Virtual Local Area Network
W-LAN	Wireless Local Area Network
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

1 Einleitung und Motivation

Die vorliegende Arbeit befasst sich mit der Konzeption, Entwicklung und Erprobung einer Monitoringumgebung für die Erfassung von Klimadaten sowie experimentellen Daten aus Messreihen von Polyamid-6 (PA6) Probekörpern. Mit Hilfe der aufgenommenen Daten soll im Rahmen von numerischen Berechnungen, unter Verwendung eines vorhandenen *Finite Elemente Methode* (FEM) Modells, das zeitlich und lokal variable Materialverhalten von PA6, unter Einfluss von Feuchtigkeit bestimmt werden. Im Kontext der Monitoringumgebung wird ein vernetztes System aufgebaut, welches die benötigten Daten für die simulationsgestützte Berechnung von Diffusionsprozessen mit zeitlich variablen Randbedingungen in Form wechselnder Umgebungsfeuchtigkeiten an PA6 Probekörpern sammelt und innerhalb des bereits am Institut verwendeten *Detact* Industrie 4.0 Systems von *Symate* (Deutschland) zugänglich macht.

Bei Polyamid handelt es sich um einen teilkristallinen Thermoplast, der auf Basis der Polykondensation synthetisiert wird [1]. Er zeichnet sich durch gute Chemikalienbeständigkeit im Kontakt mit Ölen und Schmierstoffen, hoher Festigkeit und Steifigkeit, seine gute Verarbeitbarkeit und geringe Materialkosten aus und ist insbesondere im Automotivbereich als kurzfaserverstärktes Compositematerial weit verbreitet [2, 3]. In vielen Anwendungsfällen ist der Thermoplast stark unterschiedlichen Witterungseinflüssen sowie mechanischen Belastungen ausgesetzt. PA6 besitzt die Eigenschaft bei Kontakt mit feuchter Luft oder im Direktkontakt mit Wasser Feuchtigkeit aufzunehmen und ist somit ein stark hydrophiler Kunststoff. Diese Hydrophilie führt dazu, dass PA6 bei einer Temperatur von 23°C und einer relativen Luftfeuchtigkeit von 50% beispielsweise in der Lage ist 2,5 Gew.-% Wasser aufzunehmen [2], bei einer Lagerung in Wasser sogar bis zu 10 Gew.-% [4]. In Folge der Wasseraufnahme ändern sich wichtige mechanische Eigenschaften des Werkstoffes: Unter anderem die Steifigkeit, die Festigkeit, das Volumen und die (Bruch-) Zähigkeit, da das aufgenommene Wasser als eine Art „Weichmacher“ innerhalb des Materials wirkt. Der wasserinduzierte Sorptionsprozess kann in Abhängigkeit der Umgebungsbedingungen mehrere Monate bis hin zu Jahren andauern. Hierbei gilt, je geringer die Temperatur bzw.

Umgebungsfeuchtigkeit ist, desto langsamer laufen die Diffusionsprozesse ab. [5, 6]

Aufgrund der Diffusionsprozesse ist die Wasserverteilung innerhalb des PAs zeitlich nicht konstant, so dass es durch das Vorliegen von Feuchtigkeitsgradienten innerhalb des Materials zu örtlich unterschiedlichen Materialeigenschaften durch unterschiedlichen Wassergehalt kommen kann. [7] Gerade die sich zeitlich ändernden Klimabedingungen mit unterschiedlichen Expositionen zu Medien mit unterschiedlichem Wassergehalt stellen eine Herausforderung für die technische Anwendung von PA6-Bauteilen dar [8]. Es soll mit Hilfe eines Digitalen Zwillings von PA6 Probekörpern eine Aussage über die Eigenschaften von Probekörpern mit unterschiedlichen „Lebensläufen“ getroffen werden können. Die experimentelle Bestimmung der Feuchtigkeitsverteilung innerhalb eines PA-Probekörpers ist mit teils erheblichem Aufwand verbunden [7]. Aus diesem Grund scheint eine numerische Betrachtung zur Bestimmung der entsprechenden mechanischen Kennwerte sinnvoll. Die Sorptions- und Desorptionsprozesse sind stark abhängig von den vorherrschenden Klimabedingungen und den sich daraus ergebenden Diffusionsgeschwindigkeiten, sodass eine Erfassung der Umgebungsdaten der Proben sinnvoll ist [7]. Dies umfasst unter anderem die relative Luftfeuchtigkeit, die Temperatur des Werkstoffes sowie seine Belastungsgeschichte. Die erfassten Daten sind für die weitere Verwendung in einer Simulation vorzuhalten und aufzubereiten.

1.1 Ausgangssituation für die Erstellung der Arbeit

Zu Beginn der Arbeit liegt folgende Ausgangslage vor: Es existiert ein experimentell validiertes FEM-Simulationsmodell, zur numerischen Berechnung der Wasserkonzentrationsverteilung von PA6, auf Basis der Massendiffusionsanalyse von Wasser in einem homogenen Grundmaterial. Dieses ist dazu in der Lage, nachgeschaltet an die Massendiffusionsanalyse zur Berechnung der zeitlich und lokal variablen Konzentrationsverteilungen, eine statische Spannungsanalyse mit Hilfe von Feldvariablen abzubilden [7, 9]. Innerhalb des Modells wird die Umgebungskonzentration als Randbedingung für die Massendiffusionsanalyse genutzt. Diese Randbedingungen können auch über die Zeit veränderlich definiert werden. Um die Güte des Modells mit zeitlich unterschiedlichen Konzentrationsgefällen zu erproben, sollen mit Hilfe der zu entwickelnden Monitoringumgebung solche Randbedingungen für reale Probekörper aufgenommen werden. Die erhobenen Daten sollen in dem am Institut vorhandenem *Detact* System abgelegt werden, damit sie für die Simulation abrufbar sind. Aus früheren Projekten sind *Raspberry Pi Zero W* Einplatinencomputer mit Kameramodulen vorhanden. Eine Klimadatenaufnahme soll, wenn möglich, unter

Verwendung des bereits vorhandenen Materials realisiert werden. Das FEM-Materialmodell von PA6 sowie das *Detact* System werden in Kapitel 3 „Theoretische Grundlagen“ nachfolgend beschrieben. Abbildung 1.1 zeigt eine Übersichtsdarstellung des zu entwickelnden Systems.

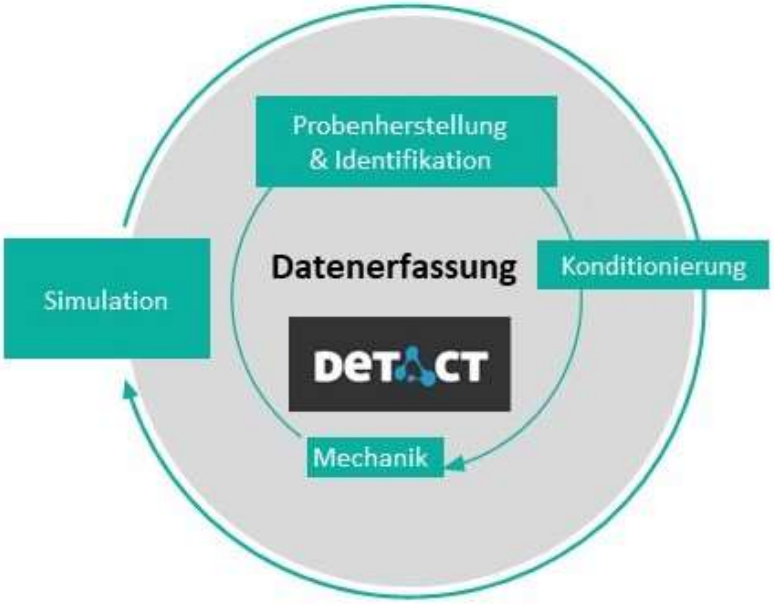


Abbildung 1.1: Systemübersicht der zu entwickelnden Monitoringumgebung [10, 11].

1.2 Konkrete Probleme und Vorgehen

Bei der Bearbeitung der Aufgabenstellung haben sich einige Themenbereiche als elementare Punkte für den Erfolg der Arbeit sowie für einen praktikablen Weiterbetrieb der im Rahmen dieser Arbeit erstellten Elemente herausgestellt. Diese werden nachfolgend aufgezählt:

- **Eindeutige Identifizierbarkeit von Probekörpern:** Die Anzahl an zu überwachenden Probekörpern, welche unterschiedliche Klimabedingungen am Institut durchlaufen, kann in naher Zukunft eine unüberschaubare Masse darstellen. Für die vollständige Erfassung der „Lebensgeschichte“ eines Probekörpers muss dieser zu jedem Zeitpunkt in seiner Historie mit geringem Aufwand identifizierbar sein. Insbesondere um die zum jeweiligem Probekörper an unterschiedlichen Stationen erhobenen Daten einfach und eindeutig zuordnen zu können. Aus diesen Gründen sind die Vergabe und das Management von eindeutigen Bezeichnern ein elementares Problem.

- **Datenerhebung von unterschiedlichen Quellen:** Die eindeutig identifizierten Probekörper sollen in der Zeitspanne ihrer Existenz verschiedene Stationen durchlaufen, an welchen unterschiedliche Klima- und verschiedenartige experimentelle Daten aufgenommen werden. Die Erstellung dieser Stationen in Soft- und Hardware bzw. Anbindung existierender Stationen an das *Detact* System ist ein weiterer grundlegender Arbeitspunkt dieser Arbeit.
- **Modellvalidierung:** Mit Hilfe, der von der Monitoringumgebung aufgenommenen Daten, soll das vorhandene und in Kapitel 3.2 beschriebene FEM-Materialmodell anhand von geeigneten Versuchsreihen validiert werden. Diese Untersuchung stellt den letzten Punkt der Bearbeitung der Aufgabenstellung dar.

2 Stand der Technik

Dieses Kapitel gibt einen Überblick über den Stand der Technik aus den für die zu entwickelnde Monitoringumgebung relevanten Bereichen. Anfänglich werden in Kapitel 2.1 die relevanten Begriffe erläutert. Anschließend werden in Kapitel 2.2 für den Anwendungsfall gebräuchliche IoT-Protokolle vorgestellt. In Kapitel 2.3 und 2.4 werden auf dem Markt verfügbare Industrie 4.0 Softwaresysteme und Sensorlösungen vorgestellt. Abschließend werden in Kapitel 2.5 die vorgestellten Elemente in ihrer Bedeutung für die Aufgabenstellung eingeordnet.

2.1 Erläuterung relevanter Begriffe

Internet of Things (IoT): Das *Internet of Things* beschriebt Systeme aus physikalischen Objekten die in Software, Netzwerke oder Elektronik eingebunden sind. Diese intelligenten Objekte sammeln und übermitteln Daten. Die Kommunikation der Systeme kann dabei auch über das Internet stattfinden. Die dem IoT-Bereich zugeordneten Objekte sind zumeist gut in die physikalische Umgebung integriert und weisen einen hohen Grad an Autonomie auf, sodass sie ihre Aufgaben oft unabhängig von anderen Elementen erfüllen können. Insbesondere ist die Kommunikation zwischen den einzelnen IoT-Geräten (auch *Machine-to-Machine* (M2M) Kommunikation) ein elementares Merkmal. [12, 13]

IoT kann als eine Weiterentwicklung von *Supervising Control and Data Accquisition* Software (*SCADA*-Software) verstanden werden, die die Steuerung und Überwachung technischer Prozesse ausführt. Am System beteiligte Komponenten sammeln Daten und geben diese an die Software weiter, welche diese weiterverarbeitet und wieder ausgibt. [14] Inzwischen gibt es bereits eine Vielzahl an auf IoT fokussierte, leichtgewichtige Protokolle mit wenig Overhead (benötigte Zusatzinformationen), wie etwa *Constrained Application Protocol* (*CoAP*), *Message Queue Telementy Transport* (*MQTT*) oder *Extensible Messaging and Presence Protocol* (*XMPP*). Klassische IoT Devices sind z.B. Smartcards, Sensoren, Tracker, Smarthome-Systeme oder *Radio-Frequency Identification* (*RFID*) Devices. [12] Eine zunehmend an Bedeutung gewinnende Eingrenzung des IoT Begriffes ist der

Begriff des *Industrial Internet of Things* (IIoT). Mit diesem Begriff wird i.d.R. IoT Technologie im industriellen Umfeld und insbesondere innerhalb eines Fertigungsumfelds beschrieben. [15]

Industrie 4.0: Der Begriff der Industrie 4.0 geht zurück auf die Forschungsunion der deutschen Bundesregierung und eine gleichnamige High Tech Strategie der Bundesregierung. Der Begriff beschreibt die Informatisierung der klassischen Industrie wie z.B. die Produktionstechnik. Ziel ist die Verschmelzung der virtuellen und der physischen Welt zu so genannten cyberphysikalischen Systemen. Der Begriff ist angelehnt an die vorangegangenen Industriellen Revolutionen, die in Abbildung 2.1 dargestellt sind. [16]

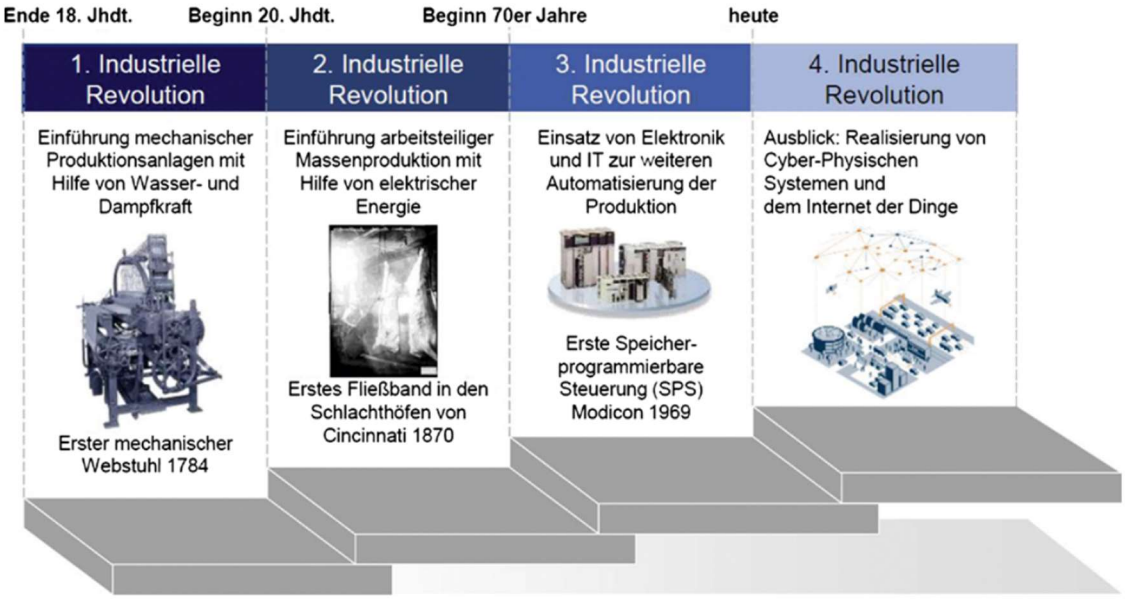


Abbildung 2.1: Stufendarstellung der unterschiedlichen Industriellen Revolutionen [16].

Im Vergleich zur dritten Industriellen Revolution steht im Kontext der vierten nicht die Verbesserung von Produktionsprozessen durch die weitläufige Verfügbarkeit von Rechenleistung im Fokus, sondern die Vernetzung unterschiedlichster Datenquellen über das Internet. Es handelt sich somit um eine weitreichende Anwendung des *Internet of Things* in einem industriellen Kontext [17, 16]. Der für den wirtschaftlichen Prozess entstehende Nutzen entspringt aus der algorithmischen Auswertung der auf diese Weise aufgenommenen Daten. Anwendungsgebiete sind dabei unter anderem das Erzielen einer flexibleren Produktion, das Schaffen einer optimierten inner- wie außerbetrieblichen Logistik, sowie die datenwissenschaftsbasierte Optimierung von Produkten oder Fertigungsprozessen.[17] Neben der Vernetzung des industriellen Umfeldes sind

auch die Selbstoptimierung und Selbstverwaltung von Industriesystemen durch Künstliche Intelligenz-Systeme (KI-Systeme) ein Ziel der Industrie 4.0. Als letzte Ausbaustufe des Industrie 4.0 Konzeptes ist eine Smart Factory zu verstehen, die weitestgehend selbstständig Maschinen steuert und intelligent auf aktuelle Rahmenbedingungen reagiert. [18, 19]

Digitaler Zwilling: Der Begriff des Digitalen Zwillings gilt als wichtiges Konzept des Industrie 4.0 Paradigmas. Er stellt eine Abbildung real physischer Gegenstände oder Systeme innerhalb des Cyberanteils cyberphysikalischer Systeme dar. Ein Digitaler Zwilling definiert sich vor allem anhand von vier Punkten [20]:

- Digitale Darstellung eines physischen Gegenstückes
- Die digitale Darstellung dient als Simulationsbasis oder ist ein Simulationsmodell
- Es existiert eine bidirektionale Datenverbindung mit dem physischen Gegenstück
- Die bidirektionale Verbindung kann über mehrere Lebensphasen des physischen Gegenstückes bestehen

Ein Digitaler Zwilling dient dazu die Leistungen, Grenzen und Potentiale einer Anlage, Maschine oder eines Bauteils zu verstehen, vorherzusagen und zu optimieren. Digitale Zwillinge bestehen zumeist aus drei Grundkomponenten, einem Datenmodell, Algorithmen und anwendungsbezogenem Wissen. Mit dem aus dem Digitalem Zwilling gewonnen Erkenntnissen lassen sich eine nahezu unbegrenzte Anzahl von Anwendungen bearbeiten. Digitale Zwillinge finden sich unter anderem in Fertigungsprozessen, im Automobilsektor, Einzelhandel, Gesundheitswesen und industriellen IoT Anwendungen wieder. [21, 22] Mit Hilfe von Digitalen Zwillingen lassen sich auf Grundlage von erhobenen Daten unterschiedliche Szenarien analysieren und deren Ergebnisse bewerten, noch bevor das real physische System in eines dieser Szenarien versetzt wird. [23] Abbildung 2.2 stellt die Wirkweise eines digitalen Zwillings dar.

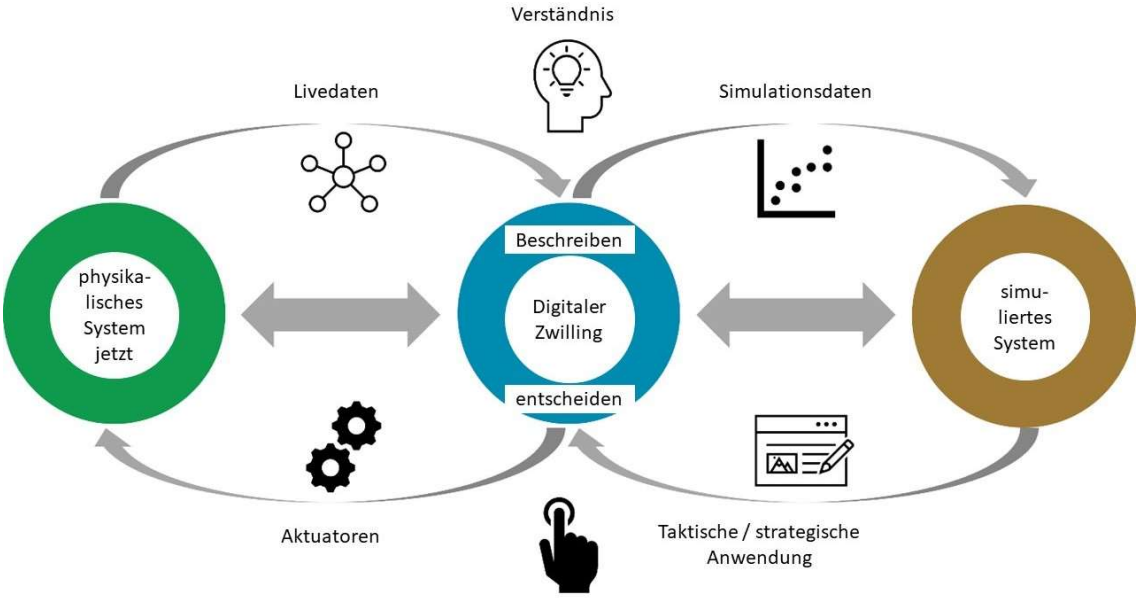


Abbildung 2.2: Schema Darstellung des Digitalen Zwillings. Die linke Seite repräsentiert den physischen aktuellen Teil des Systems, wohingegen die rechte Seite den digitalen in der Zukunft liegenden Teil des Systems repräsentiert [23].

2.2 IoT Kommunikationsstandards

Die Bedeutung von digitalen Schnittstellen zur Vernetzung von Systemen, Anlagen und Devices hat in den letzten Jahren stark zugenommen [24]. Um eine Kommunikation zwischen unterschiedlichen technischen Geräten über ein Netzwerk zu realisieren, müssen gemeinsame Standards und Regeln für die Verbindung definiert sein. Dies wird durch standardisierte Netzwerkprotokolle erreicht. Diese definieren die Handhabungen beim Senden und Empfangen von Nachrichten, das Format der Nachrichten und weitere Details zur Kommunikation. [25, 12] Protokolle lassen sich unterschiedlichen Netzwerkschichten (Layers) zuordnen. Kommunikation von Geräten auf dem gleichen Layer wird horizontale Kommunikation genannt. Die Kommunikation von Geräten auf unterschiedlichen Layern ist vertikale Kommunikation [12, 26]. Abbildung 2.3 zeigt drei für die Kommunikation über das Internet bedeutsame Schichtenreferenzmodelle im direkten Vergleich. Innerhalb dieses Kapitels werden unterschiedliche, gebräuchliche IoT Protokolle beschrieben.

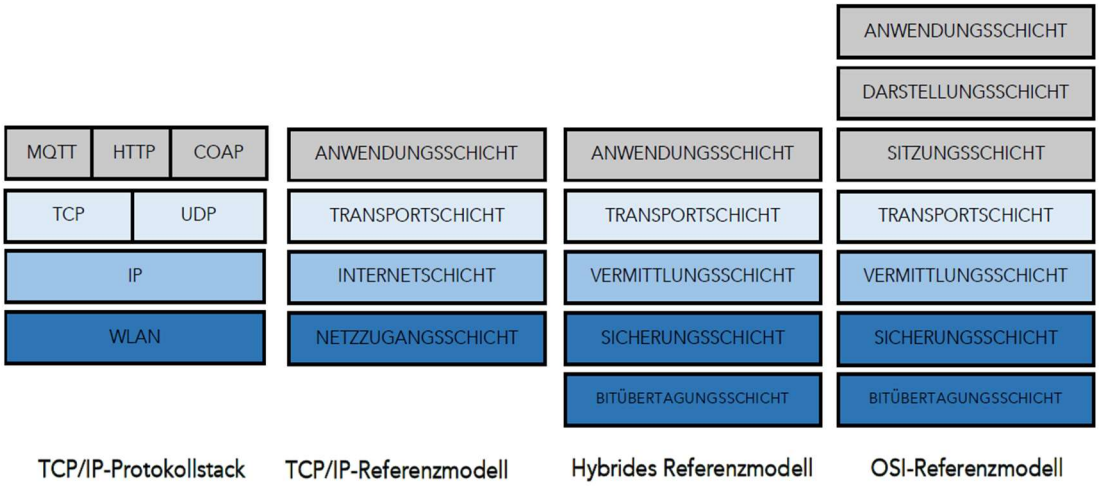


Abbildung 2.3: Vergleich der Darstellung unterschiedlicher Layer-Modelle. Im *TCP/IP*-Protokollstack für die dieser Arbeit wesentlichen Protokolle eingetragen [27].

Im Folgenden werden unterschiedliche Protokolle der Anwendungsschicht vorgestellt. Protokolle auf der Anwendungsschicht konzentrieren sich darauf Nachrichten zwischen den Schnittstellen der Anwendungen zu übermitteln. Das bekannteste Protokoll der Anwendungsschicht im Internet ist das *Hypertext Transfer Protcoll* (HTTP). Für IoT Anwendungen existieren spezielle Anforderungen, die ein IoT-Protokoll mitbringen sollte. Insbesondere ein geringer Overhead ist entscheiden, da dieser zur Entlastung der verfügbaren Netzbandbreite sowie zu geringerem Energieverbrauch der eingesetzten Devices führt. Auch ist eine hohe Übertragungssicherheit bedeutsam, da im IoT Kontext oftmals schwankende Verbindungsqualitäten mit den Netzwerken vorherrschen. [12, 26]

XMPP: Bei dem *Extensible Messaging and Presence Protocol* (XMPP) handelt es sich um einen auf dem *Extensible Markup Language* (XML) basierenden Kommunikationsstandard. Dieser wurde im Jahr 1999 von der *Jabber Open Source Community* entwickelt und ist ursprünglich für Instant Messaging Anwendungen spezifiziert. Es handelt sich um ein sicheres, offenes Nachrichtenprotokoll, welches Präsenzerkennung, Multi User Chating, Video und Sprachanrufe sowie grundsätzlich den Austausch von Daten im XML Format leistet [28, 12]. Das Protokoll unterstützt die Kommunikation mit anderen Protokollen sowie Ende-zu-Ende Verschlüsselung. Seit der zusätzlichen Spezifikation *XEP-0060* verfügt das Format auch über das Publish/Subcribe Pattern. Das Protokoll ist durchaus für IoT Anwendungen geeignet, hat aber einen vergleichsweisen hohen Overhead, wodurch es zu einer erhöhten Nutzung der Bandbreite und einen größeren

Energieaufwand kommt. [12]

HTTP / REST: Eins der bekanntesten Protokolle der Anwendungsschicht ist das *HTTP*-Protokoll, dessen Anwendung hauptsächlich aus der Übertragung von Daten für Webseiten im Internet besteht. Zu diesem Zweck senden Clients eine *GET* (empfangen von Daten), - oder *POST* (senden von Daten) *Request* an den Server, welche von diesem mit einer *Response* beantwortet wird, der ggf. die entsprechenden Daten enthält. Das *HTTP*-Protokoll basiert auf einer Client-Server-Struktur. [29, 12] *HTTP*-Nachrichten bestehen aus mehreren Teilen, dem *Header* und dem *Message Body*. Der *Header* enthält für die Übertragung von Dateien via *HTTP* wichtige Argumente und Daten wie z.B. die Sprache, den Zeichensatz oder Informationen über den Client. Der Message Body enthält die angefragten übertragenen Daten. [30] Abbildung 2.4 stellt den beschriebenen Austausch dar. Eine aus dem *HTTP*-Standard hervorgegangene Programmschnittstelle ist die *Representational State Transfere Application Programming Interface* (REST API). Es handelt sich um einen weit verbreiteten Standard für den Zugang und die Organisation von Webdiensten. *REST* bedient sich an bestehenden *HTTP*-Methoden wie *GET, POST, PUT* oder *DELETE*. Es basiert wie *HTTP* auch auf der Client-Server-Struktur. [31] HTTP-Methoden haben aufgrund der vielzähligen möglichen Argumente im *Header* einen großen Overhead [12].

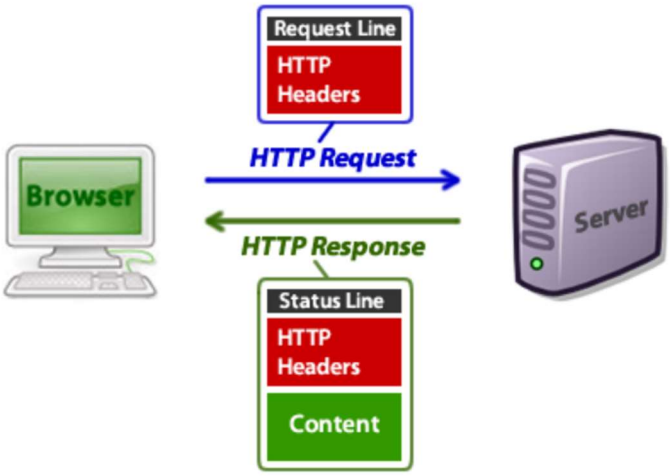


Abbildung 2.4: Ablauf der Datenanfrage und Rückgabe mit dem *HTTP*-Protokoll [30].

CoAP: Bei dem *Constrained Application Protocol* (CoAP) handelt es sich um einen auf dem *REST*-Modell basierten IoT Standard, der unter *RFC-7252* standardisiert ist. Das Protokoll kann mit *User Datagram Protocol* (UDP) oder *Transmission Control Protocol* (TCP) verwendet werden. Es nutzt die *HTTP*-Methoden *GET, POST,*

PUT und *DELETE*. Es verfügt über einen geringen Overhead und wurde für die *M2M* Kommunikation entwickelt. Durch die Unterstützung von *HTTP* kann *CoAP* auch als Web Transfer Protokoll genutzt werden. [12]

MQTT: Das *Message Queue Telemetry Transport* (MQTT) Protokoll ist ein ursprünglich von *IBM* entwickeltes Protokoll für die M2M Kommunikation. Es wurde für den Einsatz auf energiesparenden Geräten sowie an Orten mit begrenzter Netzwerkinfrastruktur entwickelt. Es weist eine geringe Komplexität sowie einen geringen Overhead auf. *MQTT* ist seit dem Jahr 2019 ein Open Source Protokoll. [32, 12]

Die Kommunikationsarchitektur von *MQTT* ist die Publish/Subscribe Architektur. Sie bietet ein skalierbares System mit hoher Flexibilität. Das Schema dieser Architektur ist in Abbildung 2.5 veranschaulicht. Die grundlegenden Elemente dieser Architektur im *MQTT*-Kontext sind ein *Broker* und Clients. Der *Broker* stellt die Hauptkomponente für die Steuerung der Verbindungen zu den Clients dar und verwaltet die globalen Daten [33]. Der *Broker* ist ein zentralisierter Server, der Daten zwischen den verschiedenen Clients vermittelt, sobald diese sich mit dem *Broker* verbunden haben. [12, 33]

Die mit dem *MQTT-Broker* verbundenen Clients können zwei verschiedene Typen aufweisen: Clients, die Daten an den *Broker* übermitteln (sog. *Publisher*) und solche die Daten vom *Broker* lesen (sog. *Subscriber*). Die Kommunikation findet dadurch statt, dass *Publisher* und *Subscriber* auf den gleichen Themenbereich (*Topic*) innerhalb des *Brokers* zugreifen. Ein *Subscriber* kann auf einem bestimmten *Topic* demnach empfangen, was ein *Publisher* auf diesem Topic schreibt. Die *Topics* auf den *MQTT-Brokern* sind hierarchisch organisiert und lassen sich mit einem *Publisher* Client individuell erstellen. D.h. ein *Topic* muss nicht erst existieren, um unter ihr auf dem *MQTT-Broker* eine Nachricht veröffentlichen zu können. Abbildung 2.6 zeigt exemplarisch den hierarchischen Aufbau der *Topics* auf einem *MQTT-Broker*.

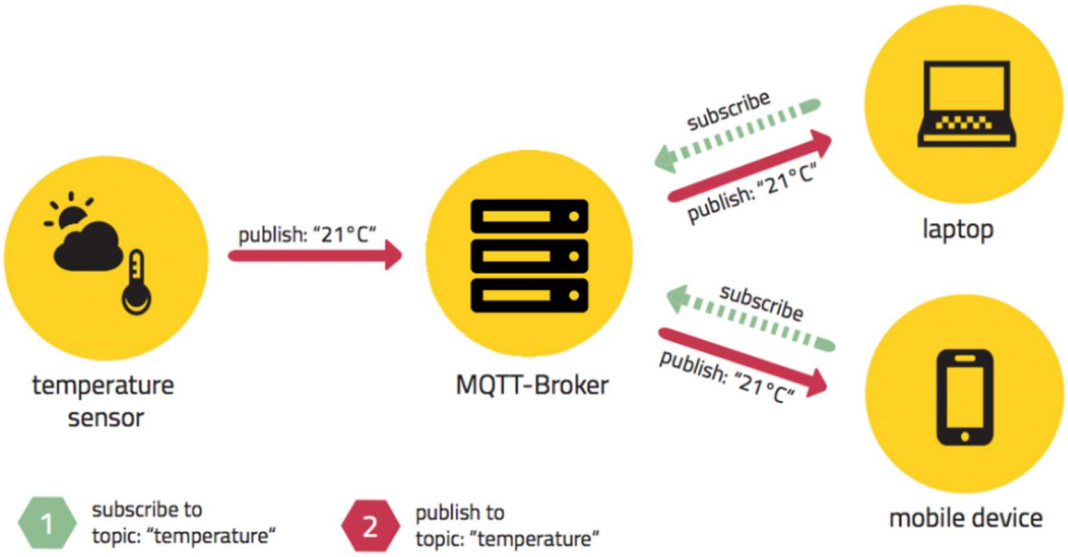


Abbildung 2.5: Darstellung der Publish/Subscribe Architektur des *MQTT*-Protokolls [33].

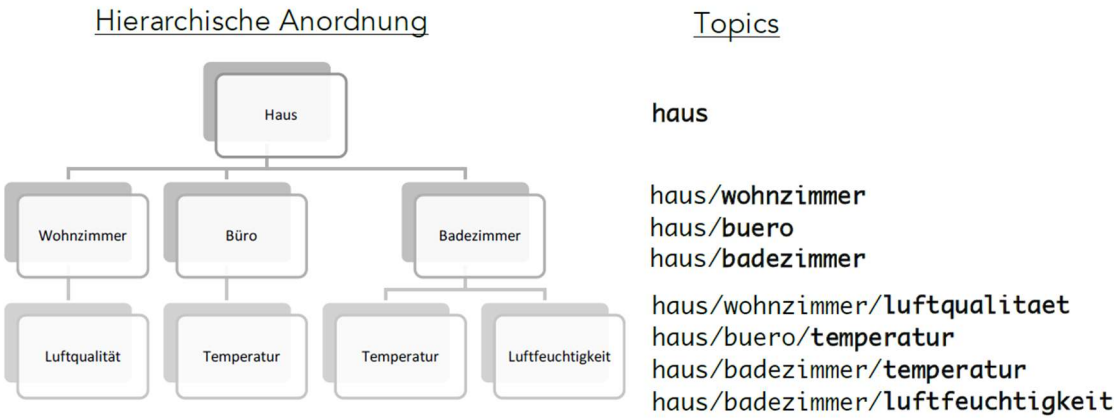


Abbildung 2.6: Darstellung des hierarchischen Aufbaus der *MQTT-Topics* am anschaulichen Beispiel einer Smarthome-Anwendung [12].

Das *MQTT*-Protokoll ist durch Setzen eines *Qualitiy of Service* (QoS) Flags in der Lage unterschiedliche Güten der Kommunikation zu erzielen. Je nach Level kann eine Garantie für die Übermittlung der Nachricht gegeben werden. Tabelle 2.1 stellt die unterschiedlichen QoS Level mit ihrer Bedeutung dar.

Tabelle 2.1: Unterschiedliche *Quality of Service* Level innerhalb des *MQTT*-Protokolls [34].

QoS Level	Art der Nachrichtenzustellung
0	„Fire and forget“: Die Nachricht wird maximal einmal zugstellt, es gibt aber keine Garantie, dass die Nachricht auch auf dem *Broker* angekommen ist.
1	Die Nachricht wird gesendet und vom Client so lange weiter übertragen bis eine Nachricht des *Brokers* an den Client den Empfang der Nachricht bestätigt. Die Nachricht wird mindestens einmal zugestellt, kann aber mehrmals zugestellt werden.
2	Die Nachricht wird genau einmal zugestellt. Es handelt sich um den zuverlässigsten, aber auch langsamsten QoS Level durch die Übermittlung der Nachricht nach dem *four-part handshake* Prinzip.

Eine weitere nützliche Funktion des *MQTT*-Protokolls ist der *Last Will and Testament* Mechanismus: Diese Funktion ist für die Möglichkeit des Verlusts von Verbindungen innerhalb von unsicheren Netzwerken vorgesehen. Sie ermöglicht bei Verlust eines Clients das Senden einer entsprechenden, vorher definierten Nachricht, auf eine bestimmte *Topic* auf den entsprechenden *Broker*. Diese Definitionen werden bereits während des Verbindungsaufbaus mit dem *Broker* getroffen. [12]

OPC UA: Der *Open Platform Communication Unified Architecture* (OPC UA) Standard von der *OPC Fundation* ist ein Standard für die M2M Kommunikation im Umfeld industrieller Automation. Der Standard ist plattformübergreifend und ursprünglich als Schnittstelle zwischen *Programmable Logic Controller* (PLC) Protokollen wie *MODBUS* oder *PROFIBUS* und *Human Maschine Interfaces* (HMI) / *SCADA* Systemen entwickelt worden. [35]

OPC UA ermöglicht die Kommunikation zwischen einer Vielzahl unterschiedlicher Komponenten eines industriellen Umfelds. So ist mit *OPC UA* die Kommunikation zwischen Maschinen, *Personal Computern* (PC), Cloud Servern und Maschinensteuerungen möglich. *OPC UA* braucht ein *IP* basiertes Netzwerk. Das Protokoll arbeitet auf der Client-Server Architektur. Dies ist in Abbildung 2.7 dargestellt. Eine Besonderheit an dem Standard besteht darin, dass der *OPC UA* Server die Kommunikation durch standardisierte Device-Modelle auf die Protokolle der jeweiligen Maschinen oder Anwendungen übersetzt. Somit ist auch eine Anbindung von anderen Protokollen auf der Anwendungsschicht mit *OPC UA* möglich. [36]

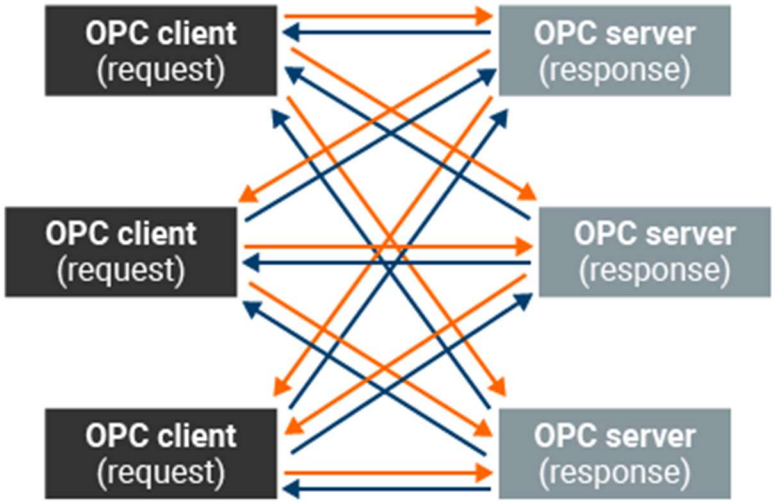


Abbildung 2.7: Darstellung der Client Server Struktur von *OPC UA* [36].

2.3 Auf dem Markt verfügbare Industrie 4.0 Systeme und IoT Systeme

Die in diesem Kapitel dargestellten, verfügbaren Lösungen sollen eine knappe Übersicht über die zum Zeitpunkt der Erstellung dieser Arbeit, zum Stand Oktober 2022 verfügbaren Industrie 4.0 Systeme geben. Es existiert eine Vielzahl Lösungen am Markt, so dass die vorgestellte Auswahl lediglich exemplarisch ist. Da die Anwendungsfälle solcher Systeme höchst unterschiedlich ausfallen können, sind auch die auf dem Markt verfügbaren Softwarelösungen zumeist auf einen hohen Grad an Individualisierung und Anwendungsfall spezifische Anpassung angewiesen, bis zu dem Punkt, an dem von Frameworks gesprochen werden muss. Es existieren gleichermaßen kommerzielle wie Open Source Lösungen. Nachfolgend werden einige Lösungen kompakt vorgestellt, die sich als IoT

Middleware, IoT Plattformen oder Datenbankverwaltungssysteme beschreiben lassen. Es handelt sich um eine Softwareebene, welche im Industrie 4.0 Kontext zwischen den IoT Geräten und den kommerziellen Anwendungen stehen oder auch solche Anwendungen zusätzlich bereitstellen. Auch das in Kapitel 2.2 beschriebene *OPC UA* kann mit unter diese Kategorisierung fallen. Der Funktionsumfang dieser Software kann stark variieren. Viele der verfügbaren Softwaresysteme decken auch mehrere Bereiche ab als es klassische Middleware tun würde. [37]

Symate Detact: Die Softwareplattform von *Symate* wird in Kapitel 3.3 beschrieben.

Eclipse Ditto: Das Open Source Framework *Ditto* von der *Eclipse* Foundation ist eine IoT Implementierung, welche einen starken Fokus auf das Managen und Bereitstellen von Digitalen Zwillingen, über gut zugängliche APIs hat. Es ermöglicht den Zugang zu Digitalen Zwillingen, in Form eines gewöhnlichen Web-Services. Auf diese Weise kann ein physikalisches Device über seinen Digitalen Zwilling wie ein Web-Service von unterschiedlichsten Programmen abgefragt werden. Es handelt sich nicht um eine allgemeine IoT Plattform, da mit *Ditto* keine Geräte angebunden oder eine Kommunikation zwischen IoT Devices ermöglicht wird. [38]

Siemens MindSphere: Es handelt sich um ein System, welches zentralisiert unterschiedliche IoT Devices anbindet und von diesen aufgenommenen Daten sammelt. Diese werden zentralisiert bereitgestellt und für die Überwachung bzw. Analyse durch Nutzer*innen oder KI aufgearbeitet. Das System ist auf individuelle Erweiterungen auch außerhalb des *Siemens* Öko Systems ausgelegt. [39]

PTC ThingWorx: Die Anwendung *ThingWorx* von *PTC* liegt in seinem Fokus auf dem IIoT Bereich. Ähnlich zu *Siemens MindSphere* und *Symate Detact* ist es als umfassende Plattform für die Anbindung unterschiedlicher industrieller Devices gedacht. Das System ermöglicht die Präsentation, Analyse und Verwaltung aufgenommener Daten. Es hat darüber hinaus Schnittstellen zu *Argumentet Reality* (AR) und *Computer Aided Desigen* (CAD). [40]

2.4 Existierende Kauflösungen für die sensorische T/H-Überwachung von Probekörpern

Temperatur- und Feuchtigkeitssensoren sind in einer großen Vielzahl am Markt verfügbar. Es handelt sich allgemein um eine etablierte Technik. Der Markt für solche Klimasensoren ist so umfänglich, dass in diesem Kapitel nicht auf einzelne

Modelle eingegangen wird, sondern lediglich die unterschiedlichen Ausbaustufen solcher Sensoren erläutert werden. Eine gebräuchliche Bauform stellt die Kombination von Temperatur und Feuchtefühler (gelegentlich auch Luftdruck) in einem Bauteil dar. Solche kombinierten Sensoren eignen sich aufgrund ihrer hohen Funktionsintegration besonders für das Erfassen von Umgebungsklimadaten. Hersteller solcher Kombisensoren sind u.a. *Bosch*, *Sensirion*, *Honeywell*, *Jumo*, *simex*, *Hengko* ect. Die Sensoren lassen sich für gewöhnlich in Chip Form oder auf Platinen verbaut mit zusätzlichen Anschlussmöglichkeiten erwerben. In dieser Form ist es nötig die Sensoren an einem mit den jeweilig nötigen Schnittstellen ausgestattetem Host System zu betreiben. Eine weitere Bauform von Klimasensoren sind Stand Alone Lösungen, welche als eigene Geräte unabhängig von einem Hostsystem betrieben werden können. Diese sind oft darüber hinaus auch in der Lage ihre erhobenen Daten über z.B. eines der in Kapitel 2.2 beschrieben Protokolle der Anwendungsschicht in einer Netzwerkumgebung weiterzugeben. Solche Sensorstationen sind als Sensoren für das direkte Verbauen auf Tragscheinen in Schaltkästen, als Outdoorwetterstationen oder als Smarthome Geräte verfügbar. Nahezu alle in großen Anzahlen erhältlichen Sensoren sind elektronische Messinstrumente. Sie arbeiten im Bereich der Temperaturmessung zumeist resistiv und bei der Messung der relativen Luftfeuchtigkeit kapazitiv oder resistiv. [41–43]

2.5 Bedeutung für die Aufgabenstellung

Die bisher in diesem Kapitel vorgestellten Elemente werden nachfolgend hinlänglich ihrer Bedeutung für das Bearbeiten der Aufgabenstellung dieser Arbeit eingeordnet. Da bereits innerhalb der Aufgabenstellung eine Anbindung der zu entwickelnden Elemente der Monitoringumgebung an das am Institut vorhandene *Detact* System der Firma *Symate* gefordert ist, haben die im Kapitel 2.3 erwähnten Industrie 4.0 Systeme anderer Hersteller keine Relevanz für die Bearbeitung der Aufgabenstellung und sind nur der Vollständigkeit halber aufgeführt. Da zum Zeitpunkt der Anfertigung dieser Arbeit keine Systeme bekannt waren, die eine Erfassung von Klimadaten und Kennungen von sich durch das betreffende Klima bewegenden Elementen auf dem Markt verfügbar ist, soll eine Anlage mit Hilfe von *Raspberry Pis* und *I2C* BUS (Inter-Integrated Circuit BUS) T/H Sensoren entwickelt werden. Eine Anlage zur Erfassung der Aufenthaltsorte der einzelnen Proben ist für die Zuordnung des Raumklimas ebenfalls zwingend erforderlich und lässt sich ggf. mit in einer solchen Sensorstation integrieren. Zur Übermittlung der Daten wird auf das in Kapitel 2.2 beschriebene *MQTT*-Protokoll

zurückgegriffen werden, da dieses explizit für IoT Anwendungen spezifiziert ist und einen entsprechenden Funktionsumfang mit sich bringt. Darüber hinaus ist das *Detact* System in der Lage über *MQTT* zu kommunizieren. Auch ist die Verbreitung von *MQTT*-Unterstützung für Python Anwendungen auf dem *Raspberry Pi* umfänglich.

3 Theoretische Grundlagen

Innerhalb dieses Kapitels werden die theoretischen Grundlagen erläutert, auf die im Rahmen der Bearbeitung der Aufgabenstellung zurückgegriffen wird. Dies ist zum Ersten das Bestimmen von Diffusionskoeffizienten für Sorptionsberechnungen an PA6 als auch das von Emde [6] und Sambale [8] entwickelte Abaqus FE-Simulationsmodell zur Berechnung der Massendiffusion und Spannungsverteilung in PA6. Anschließend wird das *Detact* System in seinen Grundzügen vorgestellt.

3.1 Bestimmung von Diffusionskoeffizienten für die Wasseraufnahme von PA6

Die Bestimmung von Diffusionskoeffizienten aus experimentellen Daten für die Wasseraufnahme von PA6 ist Gegenstand zahlreicher wissenschaftlicher Arbeiten. Die Berechnung dieser beruht auf Grundlage des Fick'schen Gesetzes aus der Anfangssteigung der Sorptionskurve. Dies wurde unter anderem in den Arbeiten von Abacha et al. [44] und Vlasveld et al. [45] experimentell an PA-Folien und an spritzgegossenen PA-Platten untersucht. Die Arbeiten untersuchen die Temperaturabhängigkeit der Diffusionskoeffizienten mit Hilfe einer modifizierten Arrhenius-Gleichung. Nach Emde [6] ergibt sich für die Bestimmung eines Diffusionskoeffizienten in Abhängigkeit des V/A Verhältnisses des Probekörpers die folgende Gleichung (3.1), mit der sich der Diffusionskoeffizient im linearen Bereich der Sorptionskurve bestimmen lässt:

$$D = \frac{\pi}{16}\left(\frac{\frac{M_t}{M_{max}}}{2\frac{V}{A_{tot}}}\right)^2 = \frac{\pi}{16} * k^2 \tag{3.1}$$

mit

D	Diffusionskoeffizient
M_t/M_{max}	Verhältnis der Masseänderungen
t	Zeit
V	Volumen
A_{tot}	Gesamte Oberfläche des Probekörpers
k	Steigung der Sorptionskurve

3.2 Abaqus FEM-Simulationsmodell für die Sorptionsrechnung von Polyamid 6

Das in dieser Arbeit verwendete FEM-Simulationsmodell für das Sorptions- und Quellverhalten von PA6 ist in der Simulationsumgebung *Abaqus (Version 2021)* von *Dassault Systèmes Simulia Corp*. (Frankreich) erstellt worden. Das FE-Modell ist ein auf experimentell ermittelten Werten basierendes Modell und somit in seiner Güte von der Qualität dieser Werte abhängig. Innerhalb von Abaqus ist es nicht möglich eine Berechnung von Sorptionsprozessen und einer Spannungsanalyse in einer einzelnen Berechnung abzubilden. Das Konzentrationsprofil eines zu berechnenden Probekörpers folgt aus der Massendiffusionsanalyse zu jedem betrachteten Zeitpunkt unter Verwendung von konzentrationsabhängigen Diffusionskoeffizienten (siehe Kapitel 3.1) und einer zeitlich veränderlichen Umgebungskonzentration. Die zeitlich veränderliche Umgebungskonzentration wird aus Messdaten vorgegeben. Das auf diese Weise berechnete Konzentrationsprofil wird mit Hilfe von Feldvariablen in die nachfolgende Simulation der statischen Spannungsanalyse des einachsigen Zugversuches übergeben. Ziel dieser Simulation ist die Vorhersage eines Elastizitätsmodules (E-Modul) für den dem Sorptionsprozessen ausgesetzten Probekörper. [8, 6] Die für den Aufbau des Modells nötigen Berechnungsformeln werden im Rahmen dieser Arbeit nicht weiter diskutiert und sind in den Werken *Experimentelle Charakterisierung und FE-Modellierung des Sorptions- und*

Quellverhaltens von Polyamid 6 in Wasser von Jessica Emde, M.Sc. [6] und *Beitrag zur Charakterisierung und Berechnung von Feuchtigkeitsverteilungen in Polyamid 6* von Anna Katharina Sambale, M.Sc. [8] nachzulesen.

3.2.1 Implementierung der Massendiffusionsanalyse

Das zeitabhängige Konzentrationsprofil des betrachteten Probekörpers folgt aus dem Modell der Massendiffusionsanalyse. Der im Rahmen der Arbeit betrachtete Mehrzweckprobekörper der Geometrie „1BA“ der DIN EN ISO 527-2 [46] wird im Kontext der Simulationsumgebung durch Ausnutzung seiner Symmetrie als Achtelmodell berechnet, was bedeutet, dass alle Abmaße aus der DIN EN ISO 527-2 des 1BA Mehrzweckprobekörpers durch zwei dividiert werden. In Abbildung 3.1 ist eine Darstellung eines vollständigen 1BA Probekörpers dem im Modell verwendeten Achtelmodell gegenübergestellt. Das Bauteil hat ein Volumen zu Oberflächenverhältnis von 0,747mm.

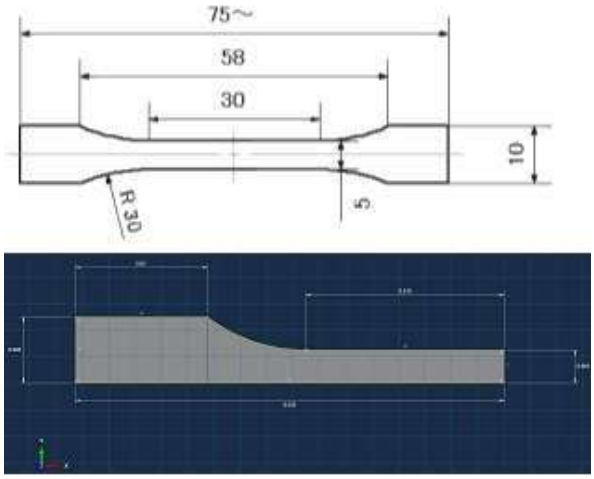


Abbildung 3.1: Gegenüberstellung einer Zeichnung eines 1BA Mehrzweckprüfkörpers zu dem innerhalb der Simulation verwendeten Achtelmodell eines 1BA Mehrzweckprüfkörpers [47, 48].

Der im Massendiffusionsmodell verwendete Netztyp ist ein Hexaedernetz des Typs (DC3D20) und besteht aus quadratischen Elementen zur Berechnung von Wärmeübergängen. Das verwendete FE-Netz muss für die Massendiffusions- als auch für die Spannungsanalyse geeignet sein, da es in beiden Modellen (mit Ausnahme der Elementtypen) übereinstimmen muss. Das Material wird über die Materialdefinition des Fick’schen Diffusionsverhaltens mit isotropen Diffusionseigenschaften definiert. Tabelle 3.1 zeigt das Modell PA6 (*Durethan B31SK,* von *LanXess*) mit Diffusionskoeffizienten bei 23°C (Raumtemperatur). Es wird eine vollständige Löslichkeit des Wassers in PA6 angenommen. Zusätzlich

wird die Definition eines Elastizitätsmodules und einer Querkontraktionszahl benötigt. Die Simulation arbeitet über die gesamte Auslagerungszeit mit konstanten Zeitschritten. Das Umgebungsmedium Wasser wird als zeitlich veränderliche Randbedingung in Form von Wasserkonzentration auf die Oberfläche der Probe aufgeprägt und ist in ppm angegeben. Die Ausgabevariablen der Massendiffusionsanalyse sind die Konzentration (*CONC*), die auf die Löslichkeit normierte Konzentration (*NNC*) und die Menge des gelösten Stoffes pro Element (*ESOL*). Die letzte Größe wird mit Hilfe eines Python Skriptes in die Intergrale Masse des im Probekörper vorhandenen Wassers umgerechnet. [48, 6, 8]

Tabelle 3.1: Diffusionsmaterialmodell PA6 für 23°C (*Durethan B31SK,* von *LanXess)* [48].

Diffusivity [m^2/s]	**[ppm]**	**Solubility [-]**	**Elastic [N/m] [-]**	**Poisson's Ration**
5.10569E-14	0	1	727000000	0,4585
1.4223E-13	30181			
4.741E-13	64557			
1.07949E-12	89741			

3.2.2 Implementierung des Zugmodells:

Die Berechnung der statischen Spannungsanalyse für den uniaxialen Zugversuch wird der Massendiffusionsanalyse nachgeschaltet, wobei sich die beiden Modelle unterscheiden. Die aus der Massendiffusionsanalyse resultierende Konzentrationsverteilung innerhalb des 1BA Mehrzweckprobekörpers wird mit Hilfe der Feldvariable „NNC" (= auf die Löslichkeit normierte Konzentration) auf das Zugmodell übertragen. Die E-Module und Querkontraktionszahlen der einzelnen Elemente können so über den jeweiligen Feldvariablenwert von der Konzentration abhängig gemacht werden [6]. Es wird ein linear elastisches Materialmodell zur Berechnung verwendet. Der Einspannungsbereich des Probekörpers wird im Modell fixiert und entlang der Längsachse des Probekörpers eine gleichbleibende Zugkraft definiert. Das Verwendete Netz ist

vom Elementtyp C3D20R „3D-Stress“, da dies für die beabsichtigte Analyseform von Nöten ist. [6, 48, 8]

3.3 Symate Detact System

Innerhalb des Institutes ist ein Industrie 4.0 System der Firma *Symate* im Einsatz. Das System wird lokal am Institut gehostet. Es verfügt innerhalb der Netzwerkinfrastruktur des Institutes über ein eigenes *Virtual Local Area Network* (V-LAN), über welches es mit den direkt an das Netzwerk angeschlossenen Geräten kommuniziert. Das System ist über ein Web-Frontend für Benutzer*innen von allen Geräten innerhalb der Institutsnetzwerkinfrastruktur zugänglich. Der Hersteller beschreibt das System als zentrales Werkzeug für die Digitalisierung in Produktionssystemen in einem Smart Factory Kontext. Das System erfasst und analysiert aus Prozessketten stammende Daten aus nahezu allen Datenquellen. Beispiele hierfür sind *OPC UA/DA*, *Euromap 63/77/15*, *QS-1-2-3-4*, *SAP*, *MQTT* etc. Abbildung 3.2 zeigt eine Prinzipdarstellung der möglichen Datenquellenbereiche, die eingebunden werden können. [11]

Abbildung 3.2: Prinzipdarstellung möglicher Datenquellen des *Detact* Systems [11].

Die erfassten Daten werden in einem vom *Detact* System verwalteten *Cassandra* Datenbanksystem abgelegt Mit Hilfe KI gestützter Analyseanwendungen ist das System in der Lage Funktionen eines *Manufacturing Execution Systems* (MES) zu übernehmen und Daten mit individuellen Zielsetzungen zu interpretieren. Durch die Zentralisierung kann das System sämtliche innerhalb der Prozesskette aufgenommene Daten zueinander in Bezug setzen und verfügbar machen. Das System visualisiert Parametereinflüsse auf Zielgrößen und kann darauf beruhend bei vielen Anwendungen unterstützend eingesetzt werden. [11]

4 Lösungskonzepte

Bei der Entwicklung von Lösungskonzepten für die Realisierung einer Monitoringumgebung gilt es mehrere Probleme simultan zu betrachten: Ein wesentliches Problem ist die Sicherstellung der eindeutigen Zuweisbarkeit von Daten zu den jeweiligen Proben. Dies muss auf Basis einer eindeutigen, nicht verwechselbaren Bezeichnung der Probekörper realisiert werden. Nachfolgend gilt es geeignete Lösungen für die Aufnahme unterschiedlicher anfallender Daten zu finden. Die erhobenen Daten müssen in einer geeigneten Form abgelegt und abrufbar vorgehalten werden. Die Lösungsansätze zur eindeutigen Zuordnung von Daten zu einzelnen Probekörpern werden in Kapitel 4.1 beschrieben, wobei eine Beschreibung zum Aufbau der Probekörperkennung in Kapitel 4.2 folgt. Kapitel 4.3 befasst sich mit dem Gesamtkonzept der Monitoring Umgebung. Die Arten der Klimamessstationen werden in den Kapiteln 4.4 und 4.5 voneinander abgegrenzt. Kapitel 4.6 befasst sich mit den Prinzipien der Kommunikation der einzelnen Stationen untereinander. Kapitel 4.7 und 4.8 befassen sich mit der Konzeptionierung der Hardware- bzw. Softwarekomponenten. Abschließend werden in Kapitel 4.7 die Möglichkeiten der Präsentation der aufgenommenen Daten behandelt.

4.1 Benennung von Proben und Stationen sowie Sicherstellung der Nachverfolgbarkeit

Innerhalb der Monitoringumgebung werden von einer Vielzahl unterschiedlicher Stationen Daten aufgenommen. Solche Stationen sind unter anderem Klimamessstationen, Verarbeitungsmaschinen oder Prüfmaschinen. Um im Nachhinein feststellen zu können, an welcher Stelle und zu welchem Zeitpunkt Daten entstanden sind, ist es notwendig jede Sensor- oder Verarbeitungsstation innerhalb der Monitoringumgebung mit einer eindeutigen Bezeichnung zu versehen und bei der Übermittlung von Daten an das Datenbanksystem mitzugeben. Eine solche Bezeichnung ist im Idealfall so aufgebaut, dass sie bereits

erkennbare Informationen über ihre Aussage darstellt und menschenlesbar ist. Die Bezeichnung einer Sensorstation oder eines Probekörpers ändert sich über die Einsatzzeit nicht und die Anzahl an Sensorstationen im Monitoringsystem ist überschaubar, weswegen es ausreichend ist, eine Station zu Beginn der Datenaufnahme bzw. Verarbeitung bei deren erstmaliger Inbetriebnahme individuell zu benennen.

Der Benennung von Probekörpern kommt im Kontext des Aufbaues einer Monitoringumgebung eine besondere Bedeutung zu: Die mit Hilfe der Sensorstationen aufgenommenen Klima- oder experimentell ermittelten Daten müssen zu jeder Zeit eindeutig einer korrespondierenden Probe zugeordnet werden können. Aus diesem Grund muss jeder Probekörper benannt und entsprechend markiert werden. Die Benennung des Probekörpers muss zur Vorbeugung von Dopplungen in der Datenbank eindeutig sein. Darüber hinaus ist es sinnvoll eine Benennungsform zu verwenden, welche von Maschinen und Menschen gleichermaßen lesbar und handhabbar ist. Die Vergabe einer solchen Bezeichnung (im Folgenden „Sample ID“ genannt) sollte unmittelbar nach der Erzeugung des Probekörpers stattfinden. Verändern sich Informationen über den Probekörper über die Dauer seiner Existenz nicht, handelt es sich um stationäre Daten. Auch diese stationären Daten zum Probekörper sollten unmittelbar nach der Erzeugung des Probekörpers mit der Sample ID in der Datenbank abgelegt werden. Dies sind zum Beispiel Informationen zum Material bzw. der Materialzusammensetzung, aus dem ein Probekörper hergestellt wird, oder welche Geometrie er hat. Probekörper können in unterschiedlichen Herstellungsversfahren und dadurch auch an verschiedenen Stellen innerhalb des Institutes entstehen. Dies birgt das Risiko, dass Probekörper mit der gleichen Bezeichnung entstehen und Sample IDs nicht mehr eindeutig sind. Es gibt unterschiedliche Möglichkeiten eine doppelte Sample ID Vergabe zu verhindern:

- Verwendung eines zentralen Services zur Vergabe von Sample IDs
- Sicherstellung von Eindeutigkeit durch Verwendung eines eindeutigen nicht wiederholbaren Merkmals innerhalb der Sample ID

Die Verwendung eines zentralen Services, welcher durch die mehrfach vorhandenen ID Vergabe Stationen angefragt wird, bedingt das dauerhafte und zuverlässige Betreiben eines solchen Services sowie die Bereitstellung einer entsprechenden Netzwerkerreichbarkeit. Bei der Verarbeitung eines nicht reproduzierbaren Merkmals innerhalb der Bezeichnung erübrigt sich eine zentrale Vergabe. Ein Beispiel für einen solchen Bezeichner ist die *Universally Unique Identifier* (UUID). Eine *UUID* ist unter anderem zeitstempelbasiert. Da eine

UUID eine 128bit Zahl ist, ist sie nicht ausreichend menschenlesbar und ggf. im alltäglichen Gebrauch, in der Praxis nicht handhabbar. [49] Die im Folgenden beschriebene Struktur der Sample IDs orientiert sich jedoch an dem Prinzip der *UUID*. Die Sample IDs sollen sowohl für einen Menschen auf einen Blick erfassbare Informationen bereitstellen als auch für evtl. manuelle Notationen handhabbar sein. Diese Anforderung bedingt, dass die Sample IDs Teils „sprechende Schlüssel" sind und daher nicht zu viele Zeichen aufweisen.

Ein Zeitstempel stellt ein Merkmal zur Erzeugung einer solchen Eindeutigkeit innerhalb der Bezeichnung dar, welcher den Zeitpunkt der Erzeugung der Sample ID markiert. Um eine Doppelbezeichnung von Proben zu vermeiden, welche gleichzeitig an unterschiedlichen Stationen entstanden sind, ist es sinnvoll ein identifizierendes Element der ID Vergabe Station der Bezeichnung hinzuzufügen. Auf diese Weise sind auch gleichzeitig entstandene Bezeichnungen unterschiedlich. Damit das bisher beschriebene Eindeutigkeitsprinzip auch bei der Verwendung mehrerer ID-Vergabestationen funktioniert, ist es von elementarer Bedeutung, dass die ID-Vergabestationen über die Information der tatsächlichen aktuellen Uhrzeit verfügen. Die Information der Uhrzeit wird im gesamten Bereich der Monitoringumgebung über das *Network Time Protocoll* (NTP) Protokoll sichergestellt. Das *NPT* v4 Protokoll ist in der Lage die aktuelle Zeit mit einer Genauigkeit von 0,23 Nanosekunden darzustellen [50].

Durch die Verwendung von Präfixen mit festgelegter Bedeutung als auch durch die Komprimierung der Information in Darstellungen niedrigerer Zeichendichte lässt sich ein Handling durch Menschen mit solchen Proben verbessern. Mit diesen beiden Mitteln können längere Bezeichnungen für Nutzer*innen handhabbarer werden. Eine Maschinenlesbarkeit der Sample ID lässt sich durch die Verwendung eines maschinenlesbaren Codes realisieren. Dazu werden in diesem Anwendungsfall QR-Codes auf Grund ihrer weiten Verbreitung, hohen möglichen Fehlertoleranz und der damit verbundenen Ausfallwahrscheinlichkeit im Fall von Beschädigungen verwendet [51, 52]. Die im Rahmen dieser Arbeit verwendeten QR-Codes haben eine Fehlertoleranz von 30%. Die Markierung der Proben lässt sich in Kombination mit einem Label realisieren, auf dem sowohl die Sample ID im Klartext als auch in Form eines QR-Codes abgebildet ist. Somit ist die Sample ID für Menschen lesbar und für Maschinen einscannbar.

4.2 Aufbau der Probekörperkennzeichnung

Die im Rahmen der Arbeit verwendete Probekörperkennzeichnung kann in Form eines Etikettes, eines Direktaufdrucks auf den Probekörper oder einer Laserbeschriftung realisiert werden und besteht aus der Sample ID in zwei unterschiedlichen Darstellungsformen wie in Abbildung 4.1 dargestellt:

1. Darstellung der Sample ID als QR-Code
2. Darstellung der Sample ID als Klartext

Abbildung 4.1: Etikett mit der Sample ID in Klartext und als QR-Code.

Die Sample ID selbst besteht aus zwei Präfixen und einem Sample ID Body, der selbst wiederum eine Zusammensetzung unterschiedlicher Informationen dargestellt. Die Präfixe und der Body sind durch einen Punkt voneinander abgegrenzt.
Abbildung 4.2 zeigt den Aufbau einer beispielhaften Sample ID:

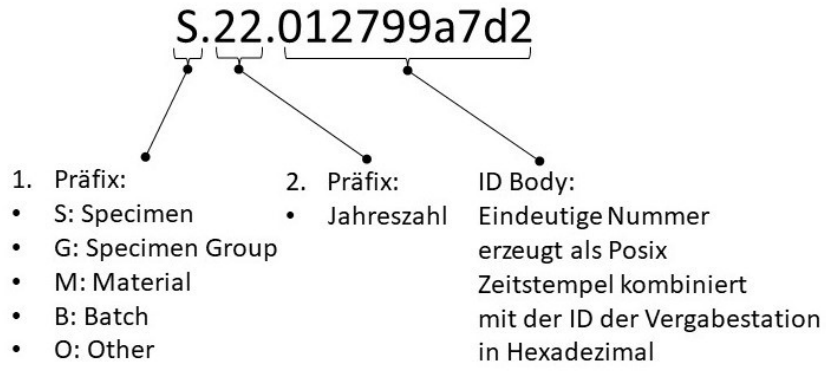


Abbildung 4.2: Aufbau der Sample ID aus Präfixen und Body.

Von links nach rechts gelesen stellt das erste Präfix eine Information über den Typ der Probe oder des Probekörpers da. Es kann fünf Zustände haben wie in Tabelle 4.1 dargestellt:

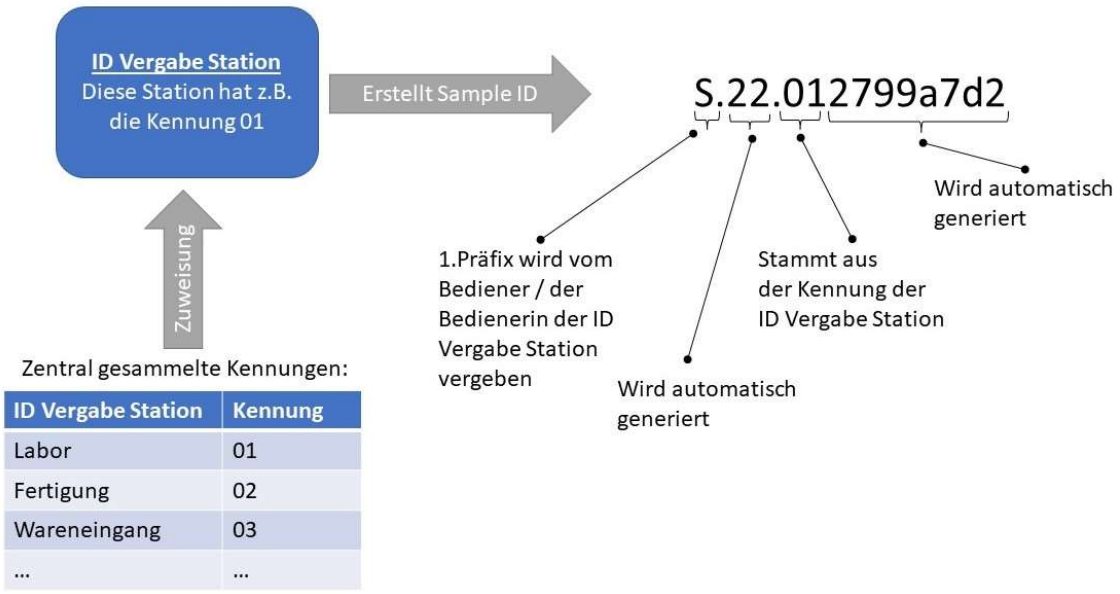
Tabelle 4.1: Bedeutung der ersten Präfixe der Sample ID

Präfix	**Bedeutung**
S	Probekörper („Sample“)
G	Probekörper Satz („Group“)
M	Material („Material“)
B	Material Charge („Batch“)
O	Ander Anwendungsfälle („Other“)

Das zweite Präfix zeichnet die Jahreszahl der Erstellung der Sample ID aus und ist für die grobe zeitliche Einordnung der Probekörper abgebildet.

Der Sample ID Body besteht aus einer Kennung der ID Vergabe Station und einem Zeitstempel. Der Zeitstempel wird im *Posix* Format (auch UNIX Zeit genannt) erzeugt. Dabei handelt es sich um die Darstellung der Zeit als die Summe der Sekunden seit dem 1. Januar 1970 0:00 Uhr UTC. Somit ist eine Zeitdarstellung in kompakter und programmatisch gut verarbeitbarer Form möglich. Da sich die ersten Stellen dieses Zeitstempels erst in 10,5 Jahren verändern wird, ist der Zeitstempel im Sample ID Body um die erste Stelle verkürzt. Damit sich der Zeitstempel um eine weitere Ziffer verkürzt, wird die sich ergebene Zahl im Hexadezimalformat dargestellt. [53]

Die jeweilige Kennung der ID Vergabe Station wird dem um eine Stelle verkürzten und hexadezimal dargestellten *Posix* Zeitstempel vorangestellt. Die Kennung der Vergabe Station ist eine fortlaufende, zweistellige, hexadezimale Nummer, die individuell bei der Einrichtung einer ID Vergabe Station vergeben wird. Es empfiehlt sich die Informationen der Kennungen der ID-Vergabestationen gesammelt aufzubewahren. Abbildung 4.3 stellt die Herkunft der unterschiedlichen Elemente der Sample ID dar.



ID Vergabe Station	Kennung
Labor	01
Fertigung	02
Wareneingang	03
...	...

Abbildung 4.3: Darstellung der Herkunft einzelner Sample ID Elemente.

Der auf diese Weise zusammengestellte Sample ID Body ist eindeutig und besteht lediglich aus zehn Stellen. Die gesamte Sample ID hat eine Länge von 15 Stellen. Dies ist eine im Ausnahmefall noch gut von Hand zu notierende Länge und bietet in dieser Hinsicht einen deutlichen Vorteil zur 128bit UUID [49]. Die Gesamtheit der Sample ID bietet durch die Präfixe die Möglichkeit, dass Nutzer*innen eine Bezeichnung sofort zuordnen können.

4.3 Allgemeine Konzeptionierung der Monitoringumgebung

Die im Rahmen der vorliegenden Arbeit aufgebaute Monitoringumgebung besteht aus mehreren Elementen:

1. Sample ID Vergabesoftware
2. Ortsfeste Klimamessstationen
3. nicht ortsfeste Klimamessstationen
4. Zwick /Roell Z2.5 Universalprüfmaschine

Mit Hilfe der Monitoringumgebung sollen Daten (z.B. Umgebungsdaten oder Prüfdaten) von Probekörpern zentral gesammelt und für FE-Simulationen vorgehalten werden.

Um eine möglichst lückenlose Überwachung der im Institut umlaufenden Proben sicherzustellen, werden unterschiedliche Messstationen entwickelt. Es werden

räumlich fest installierte Klimamessstationen als auch mobile Klimamessstationen verwendet. Darüber hinaus ist eine *Zwick/Roell ZwickiLine Z2.5* Universalprüfmaschine mit in die Monitoringumgebung eingebunden, auf der nach Abschluss der Auslagerung der Proben die mechanischen Kenndaten der jeweiligen Proben ermittelt werden. Zusätzlich gibt es ein Programm, welches auf mehreren Computern verwendet werden kann mit dessen Hilfe sich die in Kapitel 4.1 und 4.2 beschriebenen Probenkennzeichnungen erstellen und in das *Detact* System übertragen lassen.

4.4 Ortsfeste Klimamessstationen (ortsfeste Terminals)

Für die Monitoringumgebung wurden vier fest installierte Klimamessstationen fertiggestellt. Diese Stationen werden nachfolgend „ortsfeste Terminals“ genannt. Die ortsfesten Terminals sind fest an einem Ort ohne direkte Sonneneinstrahlung aufgestellt. Unter Verwendung eines Sensors (T/H Sensor) werden die Temperatur und die relative Luftfeuchtigkeit der Umgebungsluft gemessen und die dabei entstandenen Daten über eine kabelgebundene Netzwerkverbindung an das *Detact* System übermittelt. Das ortsfeste Terminal besteht aus einem Einplatinen-Computer, einem Touchscreen, einem T/H-Sensor für die Umgebungsklimadaten und einer Scaneinrichtung. Mit der Scaneinrichtung können Proben, die wie in Kapitel 4.1 und 4.2 beschrieben mit Sample IDs versehen sind, an dem ortsfesten Terminal angemeldet werden. Das Terminal überträgt dann die Klimadaten zu den angemeldeten Proben an das *Detact* System. Die Datenübertragung endet erst, wenn die Probe am Terminal wieder abgemeldet wird. Eine solche An- und Abmeldung lässt sich über die Scaneinrichtung des ortsfesten Terminals benutzerfreundlich realisieren: Die Anzahl an Proben, welche am ortsfesten Terminal angemeldet werden können, ist unbeschränkt. Zur Sicherstellung der Zuverlässigkeit der gewonnenen Daten sollten die Proben in unmittelbarer Nähe des ortsfesten Terminals schattig gelagert werden. Die ortsfesten Terminals bieten die Möglichkeit, Proben an Plätzen an denen gewohnheitsmäßig viele Proben lagern zu überwachen, ohne einen großen Mehraufwand für die Handhabung und Bereitstellung der Sensorsysteme zu betreiben. Die ortsfesten Terminals eignen sich besonders für Proben die ortsfest ausgelagert werden oder nur kurze Transferzeiten von einem überwachten Lagerort zum nächsten überwachten Lagerort haben.

4.5 Nicht ortsfeste Klimamessstationen (mobile Terminals)

Ergänzend zu den ortsfesten Terminals, werden zwei nicht ortsfeste Messtationen entwickelt und aufgebaut, die sich durch ihre deutlich kompaktere Bauweise von den ortsfesten Klimamessstationen unterscheiden (nachfolgend „mobile Terminals“ genannt). Die mobilen Terminals erfüllen messtechnisch die gleichen Aufgaben wie die in Kapitel 4.4 beschriebenen ortsfesten Terminals, sind aber für einen anderen Anwendungszweck bestimmt: Mit Hilfe der mobilen Terminals soll die Erfassung von Klima Daten über längere Transferwege der Proben realisiert werden können. Beispiele für solche Verbringungen sind z.B. der Transport innerhalb des Institutes durch unterschiedliche Räumlichkeiten als auch das Überwachen der Klimabedingungen in Räumlichkeiten, in denen kein ortsfestes Terminal zur Verfügung steht. Zu diesem Zweck verfügen die mobilen Terminals über eine kabellose Netzwerkverbindung (W-LAN) sowie ein Akkupack mit dem sie bis zu vier Stunden betrieben werden können. Durch die Verwendung eines Netzteils kann das Akkupack wieder aufgeladen werden und die mobilen Terminals auch dauerhaft betrieben werden.

4.6 Kommunikationskonzept der Monitoringumgebung mit dem Detact System

Wie in Kapitel 4.3 beschrieben, sind unterschiedliche Gerätetypen Bestandteil der Monitoringumgebung. Das *Detact* System ist in der Lage über eine Vielzahl von Schnittstellen Daten zu empfangen [11]. Beispiele hierfür sind *OPC UA/DA*, *Euromap 63/77/15, QS-1-2-3-4, SAP, MQTT* etc. [11]. Die Anbindung der innerhalb der Monitoringumgebung betriebenen Geräte an das *Detact* System erfolgt mittels des *MQTT*-Protokolls. Die Entscheidung für die Verwendung von *MQTT* erfolgte auf Grund der weiten Verbreitung, guter Unterstützung und der Verfügbarkeit auf vielen Plattformen [12]. Zu diesem Zweck gibt es am Institut einen Server auf welchem ein *MQTT-Broker* (*Eclipse Mosquitto*) betrieben wird. Eine nähere Beschreibung hierzu ist Kapitel 2.2 zu entnehmen. Die Kommunikation zwischen den einzelnen Stationen, der Monitoringumgebung und dem *Detact* System über das *MQTT*-Protokoll erfolgt über die vorhandene Netzwerkinfrastruktur des Institutes. Innerhalb der Netzwerkinfrastruktur des Institutes befindet sich ein eigens für die Kommunikation mit dem *Detact* System aufgesetztes V-LAN. Alle Geräte, die mit dem *Detact* System kommunizieren, müssen dem *Detact* V-LAN, unter Verwendung ihrer *Mandatory-Access-Control* Adresse (MAC-Adresse), zugewiesen werden. Darüber hinaus gibt es auf zwei W-LAN Accespoints eine DETACT Service Set Identifier (SSID), welche auch ins *Detact* V-LAN brückt.

Abbildung 4.4 veranschaulicht den Aufbau der Netzwerkinfrastruktur. Der *MQTT-Broker* ist mit einer Benutzerauthentifizierung durch Benutzernamen und Passwort ausgestattet, um ein versehentliches Schreiben oder eine beabsichtigte Manipulation von außen auszuschließen. Durch die Verwendung des freien Standards *MQTT* ist es möglich selbst entwickelte Devices oder ältere Messinstrumente und Anlagen individuell an das *Detact* System anzukoppeln.

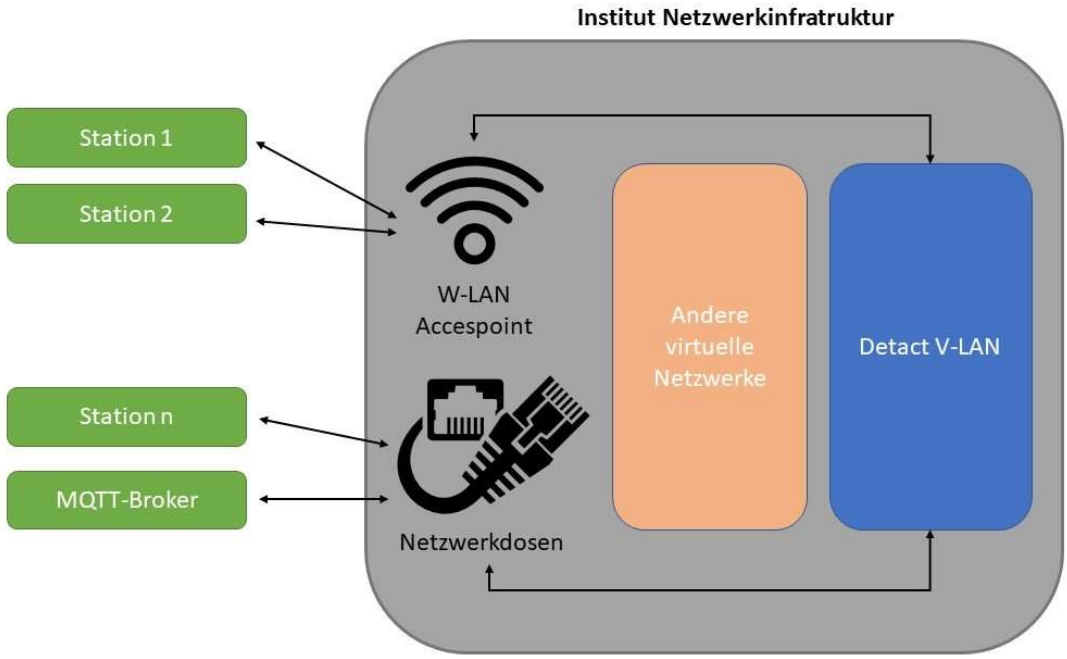


Abbildung 4.4: Darstellung der Netzwerkinfrastruktur [54, 55].

4.6.1 Form der Kommunikation über das MQTT Protocol

Die von den unterschiedlichen Stationen aufgenommenen Daten werden über *MQTT* an das *Detact* System übertragen. Bei der Übertragung der Daten veröffentlichen die einzelnen Stationen auf dem *Broker* eine Nachricht unter der *Topic* „probekoerper“. Diese Nachricht ist im *Java Script Object Notation* (JSON) Format und beinhaltet alle Daten, die an das *Detact* System übertragen werden sollen. Das JSON Format ist in nahezu allen gebräuchlichen Computersprachen etabliert und für die Übertragung von Daten über Netzwerkschnittstellen weit verbreitet [56–58]. Jede an das *Detact* System übermittelte Nachricht muss mindestens die folgenden drei Schlüssel mit entsprechenden Werten enthalten:

- **sample_id:** Benennung des Probekörpers, dem die nachfolgenden Daten zugeordnet sind.
- **station_id:** Station, an der die Probe angemeldet ist und welche die Daten

erhoben hat.

- **timestamp:** Zeitpunkt zu dem die Daten erhoben wurden im Format: DD-MM-YYYY_hh-mm-ss

Die Abbildung 4.5 zeigt den Aufbau eines *JSON*-Datensatzes der die drei notwendigen Schlüssel : Wert-Paare enthält. Die Anzahl der Schlüssel : Wert-Paare ist nicht begrenzt.

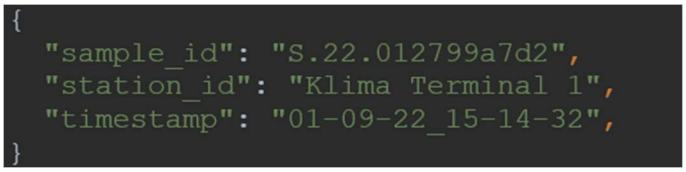


Abbildung 4.5: Darstellung der drei notwendigen Schlüssel : Wert Paare im *JSON* Format.

Das *Detact* System hat die *Topic* „probekoerper“ auf dem *Broker* abonniert und pflegt unmittelbar nach Veröffentlichung einer Nachricht der Stationen die in der Nachricht enthaltenen Daten in das System ein. Der Ablauf der beschriebenen Kommunikation ist in Abbildung 4.6 dargestellt. Die Anzahl der nach den ersten drei genannten Schlüssel : Wert-Paaren übertragenen Schlüssel : Wert -Sätze sind nicht begrenzt, frei definierbar und dadurch individuell anpassbar. Die entsprechenden übertragenen Parameter müssen dem *Detact* System allerdings bekannt sein, weswegen sich diese für die Einrichtung manuell im *Detact* System einpflegen lassen.

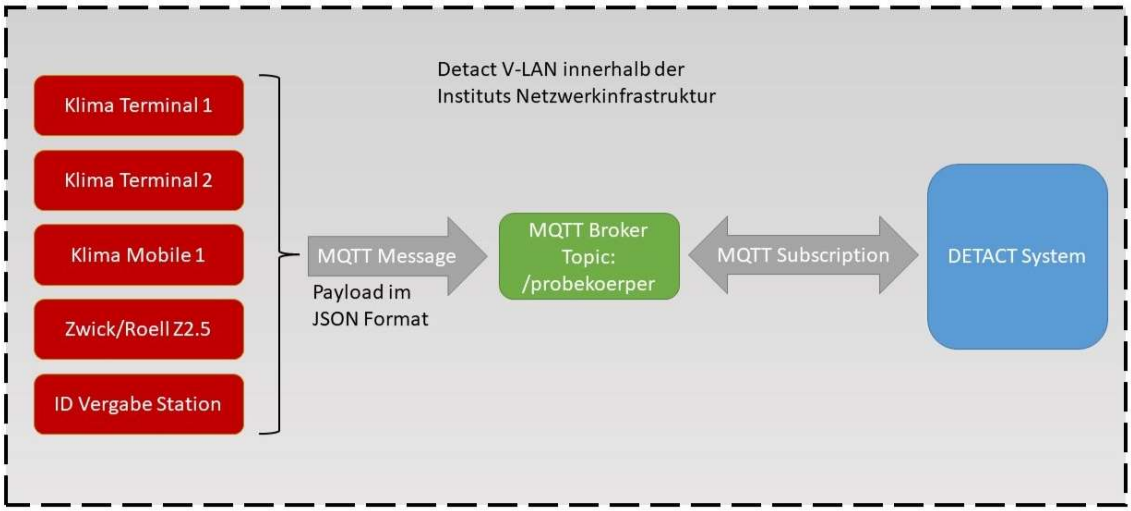


Abbildung 4.6: Darstellung der Kommunikationswege innerhalb der Monitoringumgebung.

4.7 Konzeptionierung der Hardware

Innerhalb dieses Kapitels werden die allgemeinen Ansätze für die Hardwarekonzeptionierung erläutert. Die für die Monitoringumgebung konstruierten Klimamessstationen werden auf Basis von *Raspberry Pi* Einplatinencomputern erstellt. Dies ermöglicht im Vergleich zu Kauflösungen eine hohe Variabilität des zu verwendenden Zubehörs. Darüber hinaus ist es auf diese Weise auch möglich das An- und Abmelden der Proben an den einzelnen Stationen via Scanvorgang direkt mit in die Sensorstation zu integrieren. Für die *Raspberry Pi*-Computer gibt es eine Vielzahl verfügbarer Anbauteile, eine sehr gute Dokumentation der Systeme, sowie eine große und aktive Online-Community [59]. Wenn möglich sollen Anbauteile der *Pi-Fundation* (Großbritannien) selbst genutzt werden, da bei diesen von einer gegebenen Kompatibilität ausgegangen werden kann. Darüber hinaus ist es wünschenswert möglichst viele Anbauteile auf ein *Binary Unit System* (BUS) zu legen. Im Falle des *Raspberry Pis* bietets sich auf Grund der hohen Verbreitung auch in den Zubehörteilen der *I2C* BUS an [60]. Die verbauten Hardwareteile sollen verfügbar und leicht austauschbar sein. Nachfolgend werden die Hardwareplattformen der unterschiedlichen Stationen der Monitoringumgebung spezifiziert:

- **ortsfeste Terminals:** *Raspberry Pi 4* mit Touchdisplay für das Userinterface, Kamera für das Scannen von Probenetiketten und einem T/H Sensor für die Erfassung von Temperatur und relativer Luftfeuchtigkeit. Dazu eine Vorrichtung zum Aufstellen des Gerätes auf geraden Oberflächen.

- **mobile Terminals:** *Raspberry Pi Zero W* mit 2 Zeilen Display zur Ausgabe von Benutzerinformationen, Akkupack für mobile Einsätze und einen T/H Sensor für die Erfassung von Temperatur und relativer Luftfeuchtigkeit. Für die Gewährleistung der Mobilität der Sensorstation sind die leistungsschwächeren, aber stromsparsameren, *Raspberry Pi Zeros* vorgesehen. Da deren Rechenleistung nicht ausreichend ist, um das Scannen der Probekörperetiketten durchzuführen wird das Scannen und Verwalten der angemeldeten Proben über ein Android Tablet realisiert.
- **Zwick/Roell Z2.5 Universalprüfmaschine:** Die Hardware der Maschine ist proprietär und durch den Hersteller (*Zwick/Roell*) vorgegeben. Die Maschine wird über die dazugehörige Software *Testexpert III* über einen *Windows* PC bedient. Der bereits vorhandene *Windows PC* wird mit einer zusätzlichen Software in die Monitoringumgebung eingepflegt.
- **ID Vergabe Stationen:** Die Software für die ID-Vergabe ist auf beliebigen aktuellen Windows Systemen mit *Windows 7* oder neuer verwendbar.

Die genauen Spezifikationen der Hardware und Details der entwickelten Software werden in Kapitel 5 umfänglich beschrieben.

4.8 Konzeptionierung der Software:

Für den Betrieb der Monitoringumgebung werden die einzelnen Stationen im Rahmen dieser Arbeit mit entsprechender Software ausgestattet. Bei der in diesem Kapitel beschriebenen Software handelt es sich nicht um die für den allgemeinen Betrieb der Hardware nötigen Betriebssysteme oder Dienstprogramme, sondern um die für die Erfüllung der Datenaufnahme und -übertragung benötigte, eigens angefertigte Software. Folgende Geräte werden mit eigener Software versehen:

- ortsfeste Terminals
- mobile Terminals
- für mobile Terminals benötigtes *Android* Tablet
- *Zwick/Roell* Z2.5 angeschlossener *Windows* PC (Messrechner)
- ID Vergabe Software

Auf Grund der unterschiedlichen, zu versorgenden Hardwaresysteme, bietet es

sich an die Software bereits in der Konzeptphase so auszulegen, dass sie auf unterschiedlichen Plattformen verwendbar ist. Außerdem soll die Software in einer gebräuchlichen und leicht verständlichen Sprache geschrieben sein, sodass sie auch im Nachhinein problemlos von Dritten erweitert werden kann. Um dies zu gewährleisten, wird für die Entwicklung der Software vorzugsmäßig auf *Python* zurückgegriffen. Diese Sprache wird mit Ausnahme der *Android*-App in der gesamten Software der Monitoringumgebung verwendet. Die *Android*-App ist hingegen in *Java* geschrieben. *Python* ist die zurzeit am weitesten verbreitete, interpretierte Programmiersprache, die umfassend unterstützt wird [61]. Die Software wurde so entwickelt, dass sie sich möglichst gut in vorhandene Arbeitsabläufe integrieren lässt.

4.8.1 Multithreading Ansatz

Insbesondere die für die Klimamessstationen entwickelte Software muss mehrere Aufgaben gleichzeitig erfüllen: Das beinhaltet das Erfassen der durch den Sensor gelieferten Klimadaten, das Erfassen von Proben An- und Abmeldungen sowie den Transport der Daten mittels *MQTT*. Aus diesem Grund ist die Software als Multithreadingumgebung konzipiert. Ein hardwareseitiges Multithreading ist auf den *Raspberry Pi Zero W* Einplatinencomputern auf Grund ihrer einkernigen CPU nicht möglich (siehe Anhang A5.9). Somit handelt es sich beim Betrieb der Software auf den *Raspberry PI Zero W* Computern um Softwaremultithreading. Die eingesetzten *Raspberry Pi 4* Modelle haben vierkernige CPUs und sind voll Multithreading fähig (siehe Anhang A5.1). Der Multithreadinganteil der Software wurde mit dem für *Python* verfügbaren kostenfreien, plattformübergreifenden Framework *PyQT5* realisiert. Dies ermöglicht insbesondere eine einfache Kommunikation unterschiedlicher *PyQT*-Threads untereinander, welche deutlich über die eigenen Funktionalitäten im Multithreading von *Python Thread* hinausgehen [62]. Darüber hinaus bietet das *PyQT* Framework einen umfangreichen Baukasten für das Erstellen von *Graphical User Interfaces* (GUIs), welcher durch die Software *QTDesigner* einfach zu handhaben ist [63]. Die simultan zu erfüllenden Funktionen werden in separaten Threads ausgeführt: Dazu werden diese Aufgaben in Worker-Threads ausgelagert, welche im Hintergrund laufen und gegebenenfalls Informationen an den Main-Thread weitergeben. Dies geschieht mit der hier nicht näher beschriebenen „Signals und Slots“ Architektur des *PyQt5* Frameworks [64]. Abbildung 4.7 zeigt das allgemeine Prinzip am Beispiel der ortsfesten Klimamessstation.

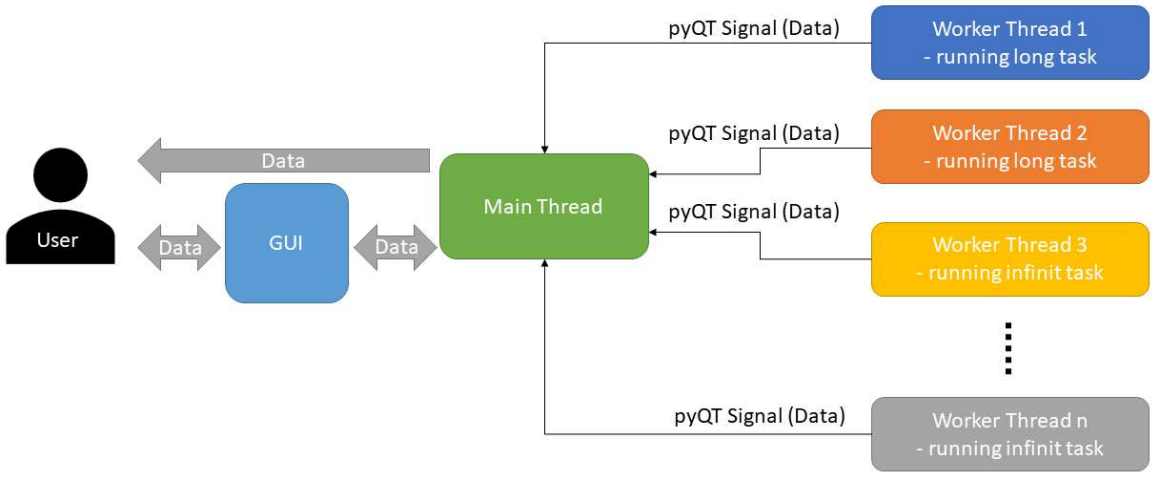


Abbildung 4.7: Darstellung des allgemeinen Multithreading Ansatzes.

Die einzelnen Threads werden aus dem Mainthread heraus gestartet. Der Main-Thread übernimmt die folgenden Aufgaben:

- Kommunikation des Programmes mit dem Benutzer / der Benutzerin entweder über eine grafische Oberfläche oder als Informationsdarstellung direkt in der Konsole.
- Starten und Beenden einzelner Worker-Threads.
- Weitergabe oder Verarbeitung von Daten die aus den Worker-Threads mit *PyQT-Signals* an den Main-Thread übermittelt werden.

4.8.2 Verarbeitung der aufgenommenen Daten

Die durch die Worker-Threads kontinuierlich aufgenommenen Daten werden innerhalb des Main-Threads verwaltet: Dazu steht ein zweidimensionales, dynamisches Array innerhalb des Main-Threads zur Verfügung. Dieses innerhalb der Software *SpecimenDataFrame* genanntes, mehrdimensionales Array besteht aus den zwei einzelnen Arrays *SpecimenNameList* und *SpecimenDataList*. Für die gleiche, laufende Variable ergibt sich daraus ein Schlüssel : Wert-Paar, welches mit jeder einzelnen Messung aktualisiert wird. Der *SpecimenDataFrame* ist demnach stets eine Repräsentation des aktuellen Status der Messstation. Auch ist es möglich dynamisch Schlüssel : Wert Paare an den *SpecimenDataFrame* anzufügen. Der Datentyp des *SpecimenNameList* Arrays ist festgelegt auf *String*. Der Datentyp des *SpecimenDataList* Arrays ist variabel, sodass er nach den Spezifikationen der abzulegenden Werte bestimmt werden kann. Die ersten drei Stellen des *SpecimenDataFrames* sind immer, wie in Kapitel 4.6.1 beschrieben, die *sample_id*, *station_id* und der *timestamp*.

Die für die *MQTT*-Kommunikation benötigten *JSON*-Strings werden aus dem *SpecimenDataFrame* heraus gebildet. Nachfolgend ist ein beispielhafter *SpecimenDataFrame* zur Veranschaulichung in Tabelle 4.2 dargestellt. Der aus dem in Tabelle 4.2 dargestellten *SpecimenDataFrame* resultierende JSON-Datensatz ist in Abbildung 4.8 dargestellt.

Tabelle 4.2: Beispielhafte Darstellung eines *SpecimenDataFrames* mit fünf Schlüssel : Wert-Paaren.

Index	**0**	**1**	**2**	**3**	**4**
SpecimenNameList	sample_id	station_id	timestamp	Temp	Humidity
SpecimenDataList	S.22.012799a7d2	Klima Terminal 1	01-09-22_15-14-32	23,35	49,66

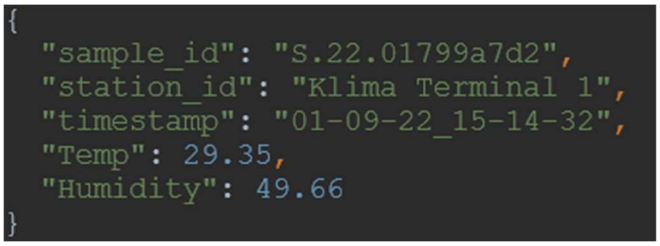


Abbildung 4.8: Darstellung des *SpecimenDataFrames* aus Tabelle 4.2 als *JSON*-Datensatz.

4.8.3 Konfiguration der Software

Da die Software ohne einen Eingriff in das Programm selbst durch Nutzer*innen oder Einrichter*innen auf unterschiedlichen Geräten konfiguriert werden können muss, ist eine Konfiguration über eine Konfigurationsdatei vorgesehen. Die Nutzer*innen müssen für die Konfiguration der Messstationen keine Kenntnisse von deren Programmierung haben. Auch bieten Konfigurationsdateien den Vorteil, dass Geräte auch über *Secure Shell* (SSH) durch Fernzugriff konfiguriert werden können.

Im Idealfall wird die Konfiguration einmalig bei der Inbetriebnahme der Messstation vorgenommen. Die Konfigurationsdatei kann dann für eine evtl. notwendige Neueinrichtung des Gerätes, an einem anderen Ort als dem Gerät selbst zusätzlich abgelegt werden. Somit kann eine Station auch von Nutzer*innen ohne Kenntnis der Konfiguration wieder in Betrieb genommen werden. Für die

Konfiguration kommen in allen innerhalb dieser Arbeit entwickelten Softwareanwendungen Initialisierungsdateien mit dem Dateiformat **.ini* zum Einsatz. Initialisierungsdateien sind Textdateien, in denen durch eine einfache Konfigurationssprache Informationen für das Programm hinterlegt sind. Die Datei selbst besteht aus unterschiedlichen Sektionen (mindestens aber einer) in denen sich Schlüssel : Wert-Paare befinden [65]. Die jeweilige Konfigurationsdatei heißt in jedem Fall „config.ini“ und befindet sich im Stammverzeichnis der Anwendung. Die Konfiguration wird beim Start der Software aus der *config.ini* herausgelesen. Die in dieser Arbeit verwendeten Konfigurationsdateien haben bis zu drei Sektionen:

- **[GENERAL]:**

  In dieser Sektion werden allgemeine Informationen über die Sensorstation hinterlegt, z.B. Name der Messstation oder auf welche Art und Weise aufgenommene Daten weitergegeben werden sollen.

- **[MQTT]:**

  Diese Sektion enthält alle nötigen Informationen, um die Verbindung mit einem vorhandenen *MQTT-Broker* aufbauen zu können, z.B. die Netzwerkadresse (IP oder Domain) oder die nötigen Zugangsdaten.

- **[DATA]:**

  Innerhalb der DATA-Sektion ist es möglich den *SpecimenDataFrame* zu definieren sowie seine initialen Werte zu hinterlegen.

4.9 Zugriff und Präsentation von Daten

Die von den Sensorstationen erhobenen Daten werden an mehreren Stellen hinterlegt und dadurch für den Benutzer / die Benutzerin sichtbar. Dies geschieht unabhängig für alle Geräte auf die folgenden Arten:

- Ausgabe der aufgenommenen Messwerte in der Anwendungskonsole.
- Ausgabe der aufgenommenen Messwerte in der GUI der Anwendung.
- Wenn entsprechend konfiguriert, erfolgt die Ablage des aktuellen *SpecimenDataFrames* in einer *.csv* Datei.
- Wenn entsprechend konfiguriert, können die übertragenen Daten auf dem *MQTT-Broker* ausgelesen werden. Dies kann mittels geeigneter *MQTT*-Software wie z.B. *MQTT Explorer* (Thomas Nordquist) erfolgen.

- Betrachtung der aufgenommenen Daten innerhalb des *Detact* Systems. Dies ist in unterschiedlicher Form unter den Punkten *Process Exploration*, *Parameter Monitoring*, *Fertigungslose* und *Metadatenanalyse* möglich. Die Abbildung 4.9 zeigt Klimadaten, welche im *Detact* System unter *Process Exploration* einsehbar sind.

Abbildung 4.9: Unter dem Reiter *Process Exploration* im *Detact* System dargestellte Klimadaten.

5 Aufbau der Monitoring Umgebung

Dieses Kapitel behandelt die Beschreibung der einzelnen, in dieser Arbeit erstellten Komponenten in Hard- und Software. Es werden alle wesentlichen Elemente in Hard und Software beschrieben. Elemente die unterstützend wirken oder marginale Aufgaben erfüllen werden nicht genauer erläutert. Diese Elemente sind in den Datenblättern und Quellcode Dateien im Anhang nachzuvollziehen. Zu Beginn werden in Kapitel 5.1 Softwareelemente vorgestellt, die in den entwickelten Komponenten mehrfach Verwendung finden. Es folgt eine Beschreibung der Klimamessstation in Kapitel 5.2 und 5.3. Kapitel 5.4 befasst sich mit der Beschreibung der in die Monitoringumgebung eingebunden Zwick/Roell Universalprüfmaschine. Abschließend wird in Kapitel 5.5 die ID Vergabe Software erläutert.

5.1 Wiederholt eingesetzte Softwareelemente

Bereits in der Konzeptionierungsphase wurde darauf geachtet, dass einmal erstellte Softwareelemente an weiteren Stationen in der Monitoringumgebung wiederverwendet werden können. Zu diesem Zweck sind mehrere *Python* Module erstellt worden. In diesem Kapitel werden zunächst alle mehrmals vorkommenden Softwareelemente in den *Python* basierten Anwendungen in ihrem Aufbau und ihrer Funktion beschrieben. Das umfasst die einzelnen Module sowie die Anweisungen des Main-Threads, welche immer nach dem Start eines Programms der Monitoringumgebung ausgeführt werden. Ein wiederkehrendes Element ist das Modul *mqtt_communicator*. Es besteht aus zwei Klassen und behandelt sowohl die Anwendungsfälle des Schreibens auf einen *MQTT-Broker* als auch den Fall des Abonnierens von *MQTT*-Topics auf einem *Broker*. Das Modul wird von allen im Folgenden beschrieben *Python* basierten Programmen importiert. In Abbildung 5.1 sind die beiden Klassen des *mqtt_communicator* Moduls beschrieben.

Das Modul *mqtt_communicator* hat die folgenden Abhängigkeiten:

- paho_mqtt [66]
- QThread [67]

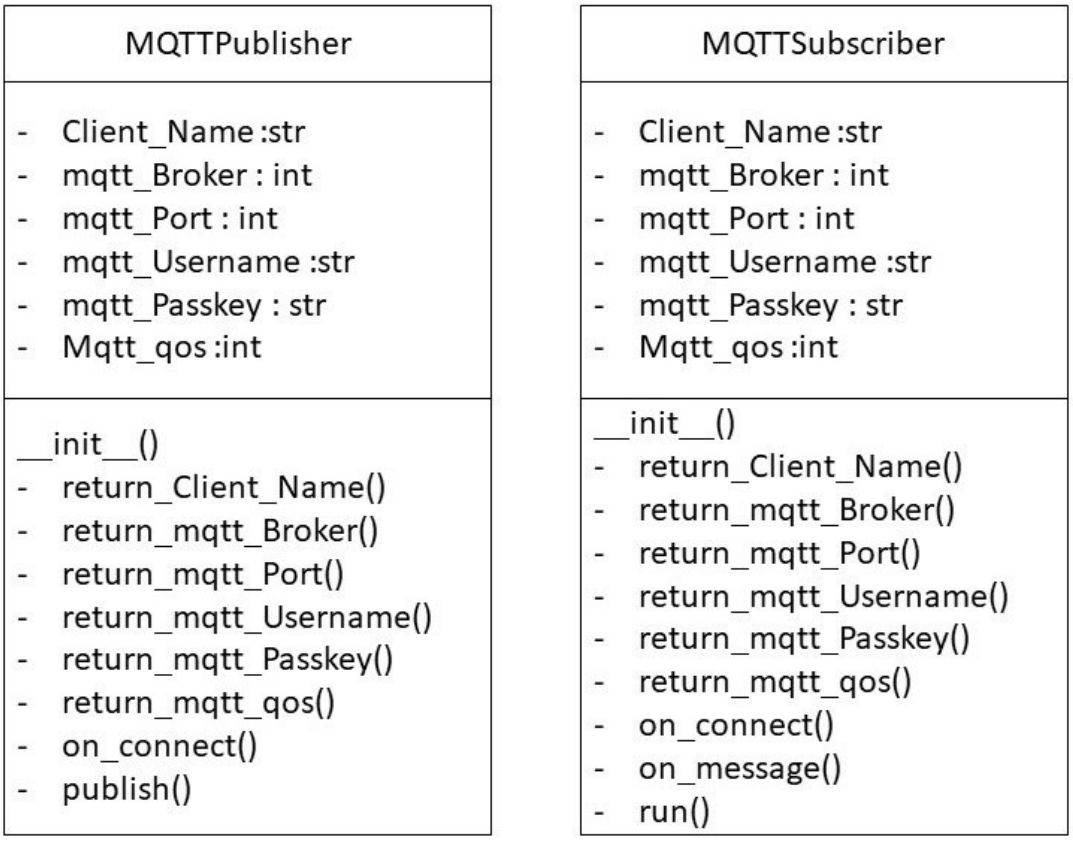


Abbildung 5.1: Aufbau der Klassen *MQTTPublisher* und *MQTTSubscriber* aus dem Modul *mqtt_communicator* als UML-Diagramm.

Mit Hilfe der *MQTTPublisher* und *MQTTSubscriber* Klassen lassen sich *MQTT*-Client Objekte erstellen, denen die in der Konfiguration hinterlegten Verbindungsdaten mitgegeben werden. Durch ihren Einsatz ist es nicht nötig die *Paho-MQTT* Definitionen stets erneut vorzunehmen.

5.1.1 Gemeinsame Elemente der Main-Threads der Software der Stationen innerhalb der Monitoringumgebung:

Alle Anwendungen der Monitoringumgebung, mit Ausnahme der *Android*-App für die mobilen Terminals, sind als Multithreadanwendungen in der Sprache *Python 3.10* ausgeführt. Bei der Initialisierung der unterschiedlichen Anwendungen werden grundsätzlich jeweils die folgenden Schritte ausgeführt:

1. **Definition des SpecimenDataFrames:**

   Der *SpecimenDataFrame* wie in Kapitel 4.8.2 beschrieben, ist das zentrale Element in allen Anwendungen der Monitoringumgebung. Er ist

Bestandteil des Main-Threads und wird durch diesen verwaltet. Andere Module haben keinen Zugriff auf die innerhalb des *SpecimenDataFrames* abgelegten Daten. Die Daten werden von anderen Threads als Signale an den Main-Thread übermittelt. Der Main-Thread pflegt auf diese Weise erhaltene Daten in den *SpecimenDataFrame* ein.

2. **Laden der benötigten Konfigurationsparameter:**

   Während der Main-Thread initialisiert, werden die benötigten Konfigurationsparameter geladen. Diese werden immer aus der sich im Stammverzeichnis der Anwendung befindlichen Konfigurationsdatei mit dem Namen „config.ini“ geladen. Für die Interpretation der Initialisierungsdatei wird das *Python* Modul *configparser* benutzt [65]. Die jeweiligen Konfigurationsparameter werden in den Anhängen A5.7, A5.16, A5.20 und A5.25 beschrieben.

3. **Ausführen von *QApplication*:**

   Um den Funktionsumfang des *PyQT5* Frameworks nutzen zu können muss innerhalb des Main-Threads ein *QApplication* oder *QCoreApplication* Objekt gestartet werden. *QApplication* wird für Programme mit GUI benötigt, *QCoreApplication* hingegen für konsolenbasierte *QT*-Anwendungen. Die *QApplication* Objekte verwalten die Ressourcen der gesamten *QT* Anwendung [68]. Auch wird die *QEventLoop* für das Handhaben von auftretenden Ereignissen durch diese Objekte ausgeführt [69]. Die Eventloop sorgt für das Erfassen von Events und das Einleiten der entsprechenden Reaktionen in anderen *QT* Objekten [70]. Sie ist somit der elementare Baustein für die Realisierung der *Signals and Slots* Architektur des *QT*-Frameworks. Ein *Signal* repräsentiert dabei ein Event und ein *Slot* stellt das Event bearbeitende Element dar [71]. Für die Kommunikation der einzelnen *QThreads* untereinander ist dieses *Signals and Slots* Prinzip elementar.

Darüber hinaus verfügt jeder Mainthread über die folgenden Methoden:

1 timestamp():

Diese Funktion generiert den Zeitstempel, der bei der Aufnahme von Daten in den *SpecimenDataFrame* mit hinterlegt wird. Der Zeitstempel wird mit dem *Python* Modul *datetime* [72] gebildet. Der erstellte Zeitstempel wird von der Funktion als String im Format *DD-MM-YYYY_hh-mm-ss* zurückgegeben.

2 build_json(Data):

Innerhalb dieser Funktion wird aus den Einträgen des *SpecimenDataFrames,* welcher der Funktion als Argument mitgegeben wird, sukzessive ein *JSON*-Datensatz (String) zusammengestellt. Zu diesem Zweck wird über die Stellen des *SpecimenDataFrames* iteriert und die entsprechenden Schlüssel : Wert-Paare abgerufen. Das Erstellen des Datensatzes (String) wird mit dem Modul *Python json* [73] erstellt. Die Funktion gibt den *JSON*-Datensatz als String zurück.

5.2 Ortsfeste Messstationen

Zum Zeitpunkt des Abschlusses dieser Arbeit wurden vier ortsfeste Klimamessstationen (ortsfeste Terminals) gebaut. Die Abbildung 5.2 zeigt ein solches ortsfestes Terminal in seiner Gesamtheit in Betrieb.

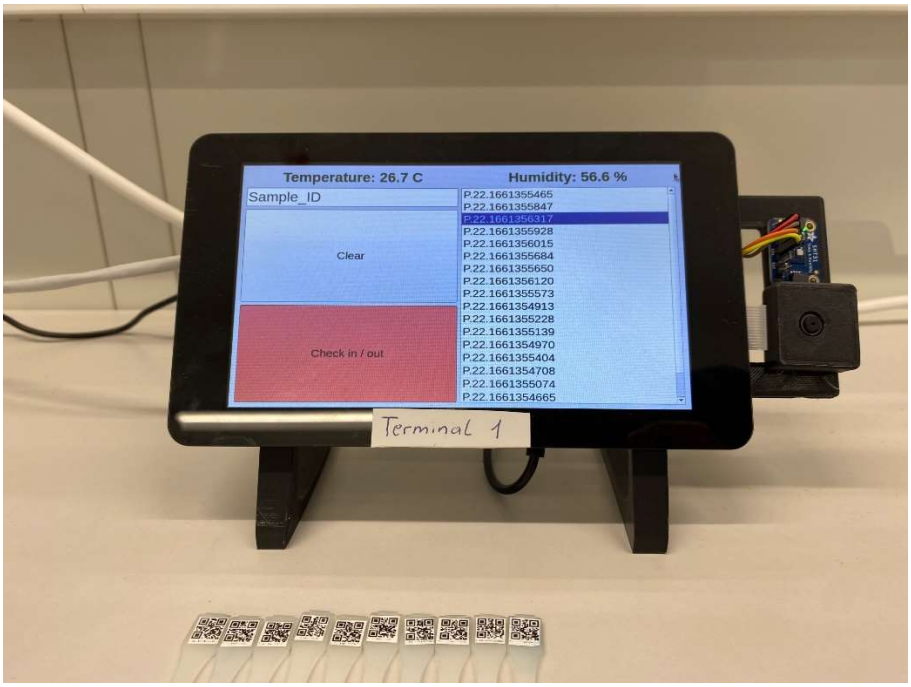


Abbildung 5.2: Ortsfestes Terminal in Betrieb. Die auf dem Display angezeigten Sample IDs wurden nach einem älteren Standard definiert.

5.2.1 Hardware der ortsfesten Terminals

Die Messstation selbst ist auf Basis eines Einplatinencomputers *Raspberry Pi 4 B* konzipiert. Dieser dient als Leistungsstarker und hinlänglich günstiger Kontroller für das Messsystem. Es handelt sich dabei um einen Einplatinencomputer mit einem vierkernigen *ARM-Cortex A72* mit einer Taktung von 1,5GHz. Die ortsfesten

Terminals werden, je nach Verfügbarkeit, mit 2GB bzw. 4GB Arbeitsspeicher eingesetzt. Der verwendete Einplatinencomputer ist für den beabsichtigten Anwendungsbereich besonders geeignet, da das verwendete *Raspberry Pi 4* Modell kompakt gebaut ist und trotzdem bereits über alle benötigten Schnittstellen wie bspw. USB-Slots, Ethernet, I2C usw. verfügt. Die bei diesem Modell zu Verfügung stehende Rechenleistung ist für den beabsichtigten Betrieb mit einer GUI ausreichend groß dimensioniert. Technische Daten sind in Anhang A5.1 hinterlegt. An den *Raspberry Pi* angeschlossen bzw. angebaut sind die folgenden Komponenten:

- **Sensirion SHT31 Temperatur- und Feuchtigkeits- (T/H-) Sensor:**

  Die Aufnahme der Klimadaten an den ortsfesten Terminals findet mit Hilfe eines *Sensirion SHT31-Sensors* (Sensirion, Schweiz) statt. Der Sensor kann die relative Luftfeuchtigkeit mit einer Genauigkeit von ±2°C erfassen [74]. Die Messungenauigkeit des Sensors der relativen Feuchte bei einer Temperatur von 25°C ist in Abbildung 5.3 dargestellt. Die Temperatur kann der Sensor mit einer Genauigkeit von ±0,3°C erfassen [74]. Die Toleranz der Erfassung der relativen Luftfeuchtigkeit liegt bei ±2% über den gesamten Temperaturbereich. Der Sensor verfügt über ein Heizelement, um die Sensoroberfläche von kondensierter Flüssigkeit zu befreien. Das Heizelement lässt sich in variablen, regelmäßigen Abständen schalten.

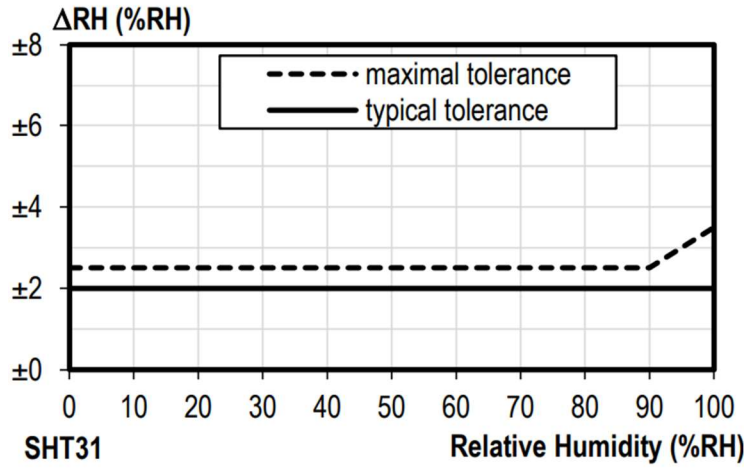


Abbildung 5.3: Darstellung des Toleranzfeldes des Temperatur- und Feuchtigkeitssensors *Sensirion SHT31* bei 25°C [74].

Der Sensor wird mittels *I2C* BUS mit dem *Raspberry Pi* verbunden. Dabei ist er innerhalb des *I2C* BUS mit der Adresse 0x44 adressiert. Die Verbindung erfolgt mit vier einadrigen Kabeln an den *General Purpose Input/Outout-Pins* (GPIO-Pins) der

Raspberry Pis. Die Sensorplatine ist von der Rückseite des *Raspberry Pis* entfernt auf eine im Folgenden näher beschriebene Trägerplatte geklebt. Das ermöglicht das Anbringen des Sensors entfernt von den Kühlkörpern des *Raspberry Pis,* um einer Verfälschung der Messergebnisse durch Abwärme des *Raspberry Pis* vorzubeugen. Der auf die Trägerplatte geklebte Sensor ist in Abbildung 5.4 zu sehen.

Abbildung 5.4: Sich in Betrieb befindlicher *Sensirion SHT31,* aufgeklebt auf Trägerplatte.

Für den *Sensirion SHT31* ist ein entsprechendes *Python* Modul zur Verwendung auf *Raspberry Pi* Einplatinencomputern verfügbar. Technische Daten des Sensors sind in Anhang A5.2 hinterlegt.

- **Raspberry Pi Kameramodule v2.1:**

  Das vom Hersteller der *Raspberry Pis* entwickelte Kameramodul, für den *Raspberry Pi* in der Version 2.1, hat eine 8 Megapixel starke Kamera mit einer Fotoauflösung von maximal 3280 x 2464 Pixeln. Videoaufnahmen sind in 1080p bei 30 FPS und 720p bei 60FPS möglich [75]. Die Kamera wird an den *Raspberry Pi* mittels der *Camera Serial Interface* (CSI) - Schnittstelle mit einem 30cm langen, flachen *CSI*-Kabel angeschlossen. Die Kamera ist nachfolgend beschriebenen kompakten Kameragehäuse eingebaut. In diesem ist die Kamera zusammen mit dem zuvor beschrieben *Sensirion SHT31* T/H-Sensor auf der Sensorträgerplatte aufgeklebt. Mit Hilfe der Kamera werden die zuvor auf den Probekörpern angebrachten, als QR-Codes codierten, Sample IDs erfasst. Für die Verwendung der Kamera stellt die *Raspberry Pi Fundation* ein entsprechendes *Python* Modul zur Verfügung. Abbildung 5.5 zeigt das Kameramodul v2.1. Technische Daten des Kameramoduls sind in Anhang

A5.3 hinterlegt.

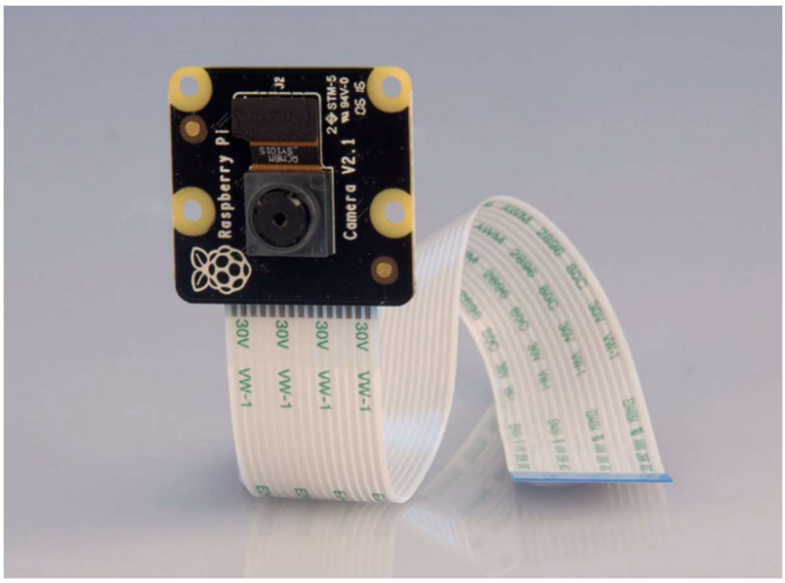


Abbildung 5.5: *Raspberry Pi Kameramodul v2.1* mit angeschlossenem CSI Flachbandkabel [75].

- **Raspberry Pi 7“ Touchscreen:**

  Das verwendete, kapazitive Touchdisplay der Raspberry *Pi Fundation* hat eine Bildschirmdiagonale von 7“ und eine Auflösung von 800x480 Pixeln [76]. Es unterstützt 10 Finger Multitouch [76]. Das Display wir mit Hilfe einer Adapterplatine mit dem *Raspberry Pi* verbunden. Es benötigt dazu zwei einadrige Kabel, mit denen 5V von den *GPIO*-Pins des *Raspberry Pis* abgenommen werden sowie ein Flachbandkabel, mit dem es an der *Display Serial Interface* (DSI)- Schnittstelle des *Raspberry Pis* angeschlossen wird. Das Display stellt die Hauptschnittstelle zum Benutzer / zur Benutzerin dar. Abbildung 5.6 zeigt das Display mit der Adapterplatine. Technische Daten sind in Anhang A5.4 hinterlegt.

Abbildung 5.6: *Raspberry Pi* 7“ Touchdisplay mit Adapterplatine, Befestigungsmaterial, DSI Kabel und Stromkabeln [76].

- **Stellfüße:**

  Direkt mit dem Display werden zwei Stellfüße verschraubt. Mit diesen lässt sich das vollständig montierte, ortsfeste Terminal auf ebenen Flächen so abstellen, dass das Display für einen guten Betrachtungswinkel leicht geneigt ist. Die Stellfüße sind als *Fused Deposition Modeling*- (FDM)-Druckteile ausgeführt und aus PLA hergestellt. Die Stellfüße sind als OpenSource von der Plattform *Thingiverse* entnommen [77]. Abbildung 5.7 zeigt ein 3D Modell eines der Füße.

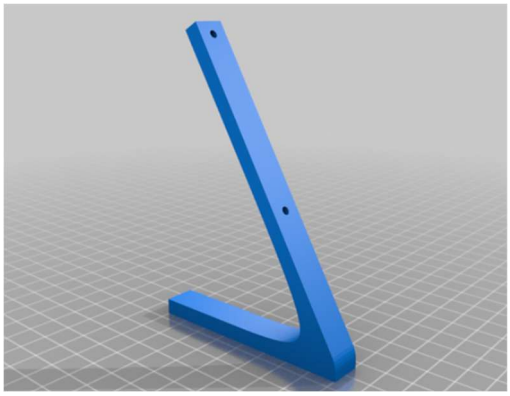


Abbildung 5.7: 3D Modell des Stellfußes. Es sind zwei Stellfüße pro Messstation verbaut [77].

- **Kameragehäuse:**
  Das Gehäuse für das Kameramodul ist ein FDM-Druckteil aus PLA. Die Bauteilmaße sind 29mm x 30mm x 12mm (Höhe, Breite, Tiefe). Die Abmessungen fassen die Kamera kompakt ein. Es besteht aus einem Grundkörper und einem Deckel. Die Verbindung zwischen den beiden Teilen wird durch Schnappharken hergestellt. Das Gehäuse stammt aus der FDM-Druck OpenSource-Datenbank *Thingiverse* [78]. Abbildung 5.8 zeigt das zusammengesetzte Gehäuse mit eingebautem Kameramodul.

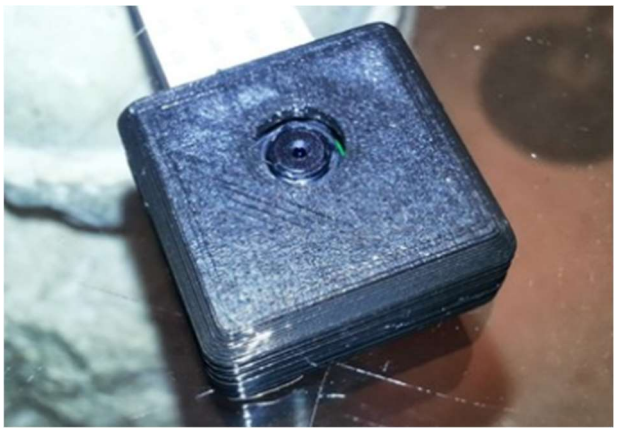


Abbildung 5.8: Zusammengesetzes Kameragehäuse mit eingebautem Kameramodul [78].

- **Sensorträgerplatte:**

  Die Sensorträgerplatte dient zur Befestigung des Kameramoduls sowie zur Befestigung des T/H-Sensors. Mit ihrer Hilfe wird die Kamera in eine Position gebracht, welche dem Benutzer / der Benutzerin zugewandt ist sowie den Sensor von den Kühlkörpern des *Raspberrys Pis* entfernt. Die Trägerplatte ist zusammen mit einem der Stellfüße am Display verschraubt. Die Trägerplatte ist als FDM-Druckteil aus PLA ausgeführt. In Anhang A5.5 befindet sich eine entsprechende technische Zeichnung.

- **Raspberry Pi USB-C Steckernetzteil:**

  Das Steckernetzteil der *Raspberry Pi Fundation* ist explizite für den Einsatz mit dem *Raspberry Pi 4 Model B* vorgesehen. Es ist für den Anschluss an 230V AC mit 50Hz Stromnetzte bestimmt und liefert 5,1V DC bei 3A. Zudem liefert es eine Leistung von 15,3W. Technische Daten sind in Anhang A5.6 hinterleg.

5.2.2 Software der ortsfesten Terminals

Die in den ortsfesten Terminals verbauten *Raspberry Pi 4 Model B* Einplatinencomputer werden mit dem generischen Betriebssystem der *Raspberry Pi Fundation*, *Raspberry Pi OS* in der Version 11 mit Codenamen „*Bullsey*“ in der 32bit Variante betrieben. *Raspberry Pi OS* ist ein *Debian* Deviat [79]. Auf den ortsfesten Terminals ist *Python 3.10* installiert und die im Folgenden beschriebene Software ist *Python 3.10* basiert. Die Beschreibung der graphischen Oberfläche der Anwendung, sowie die Beschreibung der Konfiguration mit Hilfe der config.ini Initialisierungsdatei ist in Anhang 5.7 zu finden.

Die Software, mit der die ortsfesten Terminals betrieben werden, ist, wie in Kapitel 4.8.1 beschrieben, als Multithreadumgebung im *PyQT5*-Framework ausgelegt. Abbildung 5.9 zeigt die Multithread Struktur der *Probenmonitoring* genannten Software. Darüber hinaus sind die von den Threads zu dem Main-Thread verbundenen Signals in Form von Pfeilen eingezeichnet. Nachfolgend werden die einzelnen Threads detailliert beschrieben.

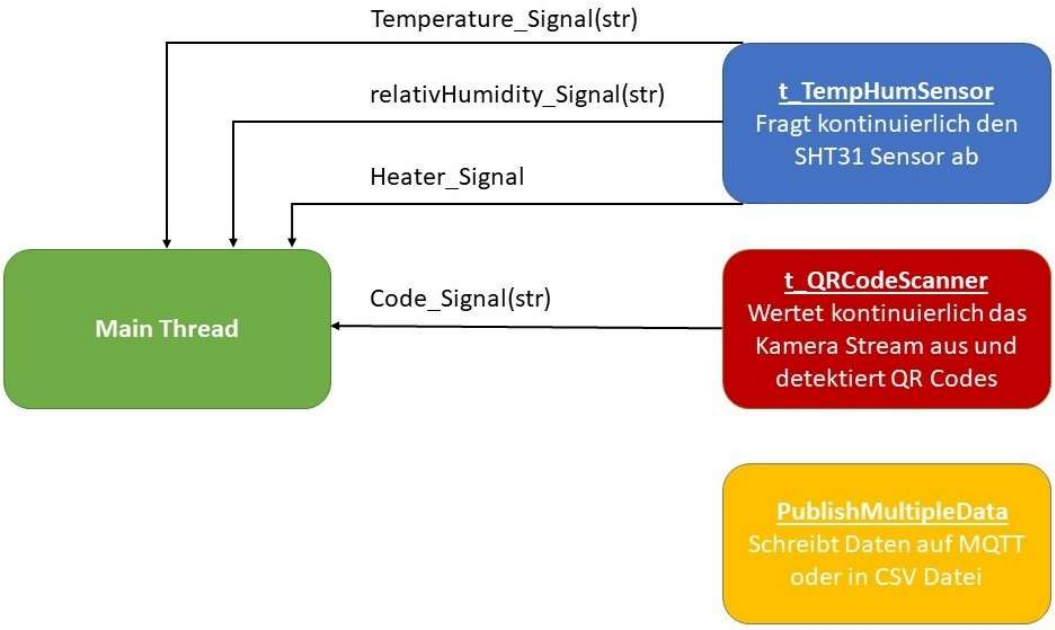


Abbildung 5.9: Übersicht über die innerhalb von Probenmonitoring eingesetzten Threads mit deren Signals.

Main-Thread: Neben den in Kapitel 5.1.1 genannten Funktionalitäten stellt der Main-Thread die folgenden Funktionen bereit:

- Starten der Worker-Threads.
- Initialisierung der GUI (siehe Abbildung 5.2)

- An- und Abmeldung von Proben. Der Prozess des Anmeldens einer Probe wird durch das Hinzufügen einer sich im Textfeld *Sample_ID_Edit* befindlichen Sample ID realisiert. Eine Sample ID, welche sich bereits in dem Array befindet, wird aus diesem entfernt.
- Slots zu den ankommenden Signals aus den Worker-Threads. Die Slots sind in Tabelle 5.1 beschrieben.

Tabelle 5.1: Darstellung der verschiedenen Signals und ihren im Main-Thread der ortsfesten Terminals hervorgerufenen Handlungen.

Signal	**Slot**
Temperature_Signal(str)	• Ausgabe des mit dem Signal übermittelten Wertes in der Konsole. • Ergänzung / Aktualisierung des *SpecimenDataFrames* mit dem im Signal übermittelten Wert.
relativ_Humidity_Signal(str)	• Ausgabe des mit dem Signal übermittelten Wertes in der Konsole. • Ergänzung / Aktualisierung des *SpecimenDataFrames* mit dem im Signal übermittelten Wert.
Heater_Signal	• Ausgabe des mit dem Signal übermittelten Wertes in der Konsole.
Code_Signal(str)	• Schreiben der mit dem Signal übermittelten Sample ID in das Textfeld *Sample_ID_Edit*.

Der Main-Thread hat die folgenden Abhängigkeiten:

- TempHumSensor aus t_TempHumSensor
- MQTTPublischer aus t_mqtt_communicator
- QrCodeScanner aus t_QrCodesanner
- datetime [72]
- random [80]
- logging [81]
- configparser [65]
- sys [82]
- csv [83]
- json [73]
- Pyqt5.QtCore [84] (Mehrere Elemente)
- QTGui [85]
- Pyqt5.Qwidgets [86] (Mehrere Elemente)

t_TempHumSensor: Innerhalb dieses Worker-Threads wird über den *I2C* BUS der in Kapitel 5.2.1 beschriebene T/H-Sensor *SHT31* ausgelesen. *T_tempHumSensor* ist als eigenes Modul definiert. Die Abhängigkeiten des Modules sind nachfolgend aufgezählt:

- QThread [67], pyqtsignal [87], pyqtslot [88]
- time [89]
- board [90]
- adafruit_sht31d [91]

Das Modul enthält lediglich eine Klasse namens *TempHumSensor*, welche in Abbildung 5.10 dargestellt ist. Die Klasse erbt aus der *QThread*-Klasse [62].

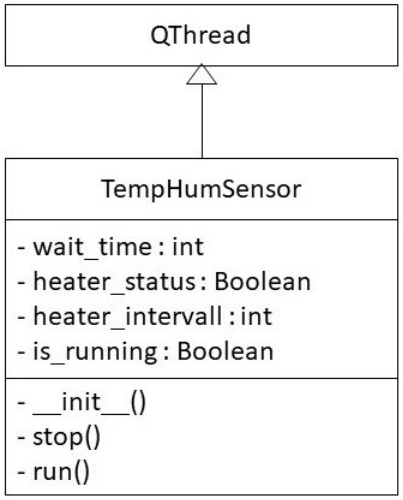


Abbildung 5.10: UML-Darstellung der Klasse *TempHumSensor*.

In der Initialisierung der Klasse wird eine Verbindung über *I2C* mit dem *SHT31* T/H- Sensor aufgebaut. Die Hauptmethode der Klasse ist die *run()*-Methode. Innerhalb dieser Methode wird in einer Endlosschleife der Sensor abgefragt, das Heizelement des Sensors gesteuert und die entsprechenden Signals emittiert. Die Methode pausiert für die Dauer der übergebenen *wait_time*. Abbildung 5.11 zeigt den Aufbau der Methode. Die Methode *stop()* unterbricht die Endlosschleife der Methode *run()* .

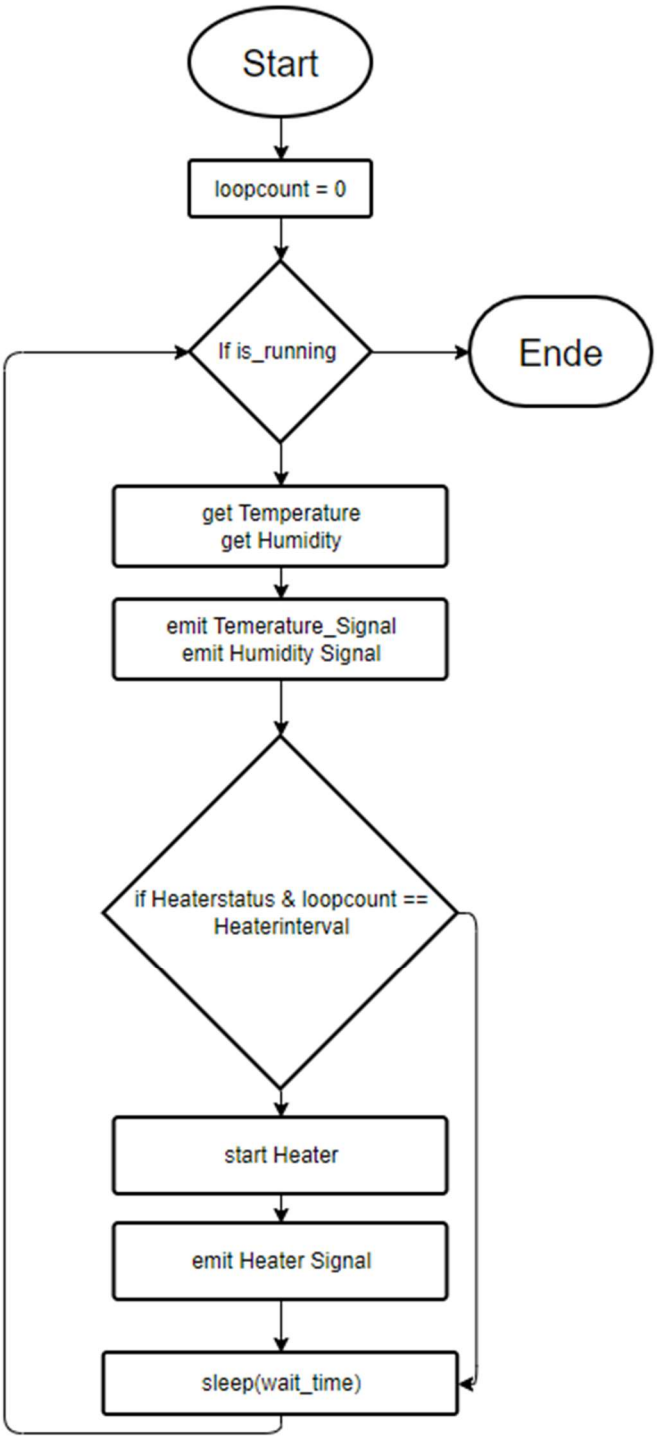


Abbildung 5.11: Darstellung der *run()* Methode der *TempHumSensor*-Klasse als Flussdiagramm.

Ebenfalls ist ein *t_TempHumSensor_bme280* Modul vorhanden. Dieses ist für die Verwendung der Messstation mit einem *Bosch bme280* Temperatur- und Feuchtigkeitssensor vorgesehen. Das Modul erfüllt die gleiche Aufgabe wie das *t_TempHumSensor* Modul und ist mit dem Sensor zusammen austauschbar.

t_QrCodeScanner: Mit Hilfe des Worker-Threads *t_QrCodeScanner* wird kontinuierlich das Kamerabild ausgewertet und auf sichtbare QR-Codes überprüft. Die Klasse *QrCodeScanner* erbt von der Klasse *QThread* [62]. Abbildung 5.12 stellt die Klasse dar. Die Klasse ist in einem Modul definiert und hat die folgenden Abhängigkeiten:

- QThread [67], pyqtsignal [87], pyqtslot [88]
- time [72]
- cv2 [92]
- pyzbar [93]
- imutils [94]

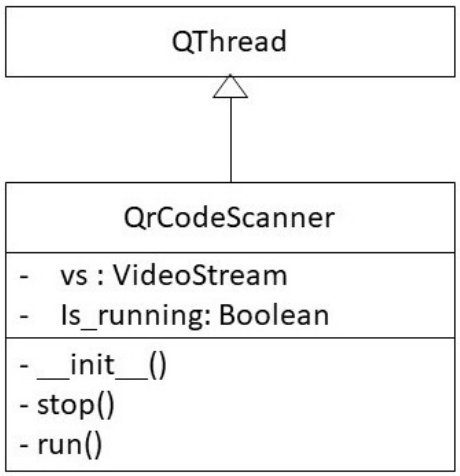


Abbildung 5.12: Die Klasse *QrCodeSanner* als UML-Diagramm.

In der Initialisierung der Klasse wird der Videostream von dem *Raspberry Pi* Kameramodul abgefragt. Innerhalb der Methode *run()* wird der Kamerastream in einer Endlosschleife pro Frame nach Strichcodes durchsucht. Dies beinhaltet Barcodes, Data Matrix Codes und QR-Codes. Abbildung 5.13 zeigt die Methode *run()* als Flussdiagramm. Die Methode *stop()* unterbricht die Endlosschleife der Methode *run()*.

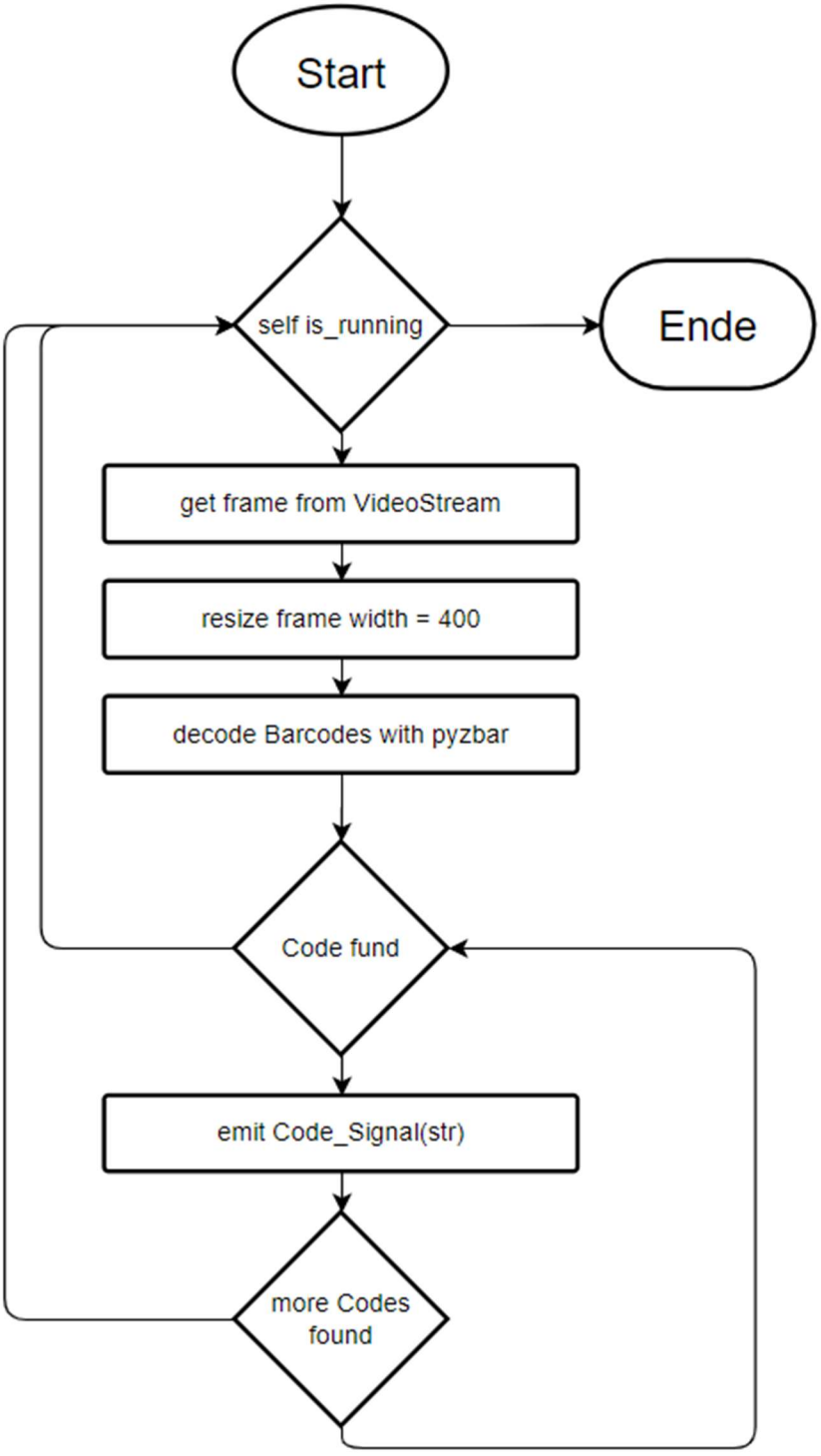


Abbildung 5.13: Die Methode *run()* der *QrCodeScanner* Klasse dargestellt als Flussdiagramm.

PublishMultipleData: Die Klasse *PublishMultipleData* ist für die Ablage der aufgenommenen Daten im *SpecimenDataFrame* mit den zugehörigen Sample IDs zuständig. Die Ablage der Daten erfolgt als Ablage des *SpecimenDataFrames* in einer CSV-Datei, als Veröffentlichung einer *JSON* formatierten *MQTT*-Message oder beidem gleichzeitig. Da dieser Worker-Thread den *SpecimenDataFrame* lesen können muss, ist er nicht als eigenes Modul definiert. Seine Klassendefinition findet sich in der *main.py*, so dass sich der Mainthread und der *PublishMultipleData* Thread einen Namespace teilen. Die Methoden zum Schreiben von CSV-Dateien sind ebenfalls in der *main.py* definiert. Die Methoden für die Veröffentlichung von *MQTT*-Messages befinden sich in einem eigenen Modul namens *mqtt_Communicator* (siehe Kapitel 5.1.1). Für die Veröffentlichung der Daten über *MQTT* weist die *run()* Methode nacheinander dem *sample_id* Schlüssel des *SpecimenDataFrames* ein Element des *Sample_Ids* Arrays zu. Dann wird aus dem *SpecimenDataFrame* mit der *build_json* Methode (siehe Kapitel 5.1.1) ein *JSON* String erstellt. Die Daten werden dann mit der *MQTT_Publisher*-Klasse (siehe Kapitel 5.1.1) auf den *MQTT-Broker* geschrieben. Anschließend wird der *SpecimenDataFrame* in eine CSV-Datei mit dem Namen der gerade bearbeiteten Sample ID geschrieben. Abbildung 5.15 zeigt die Klasse *PublishMultipleData* in ihrem Aufbau.

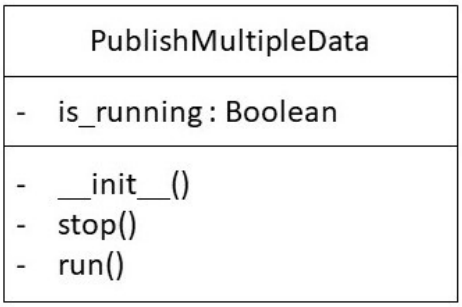


Abbildung 5.14: Darstellung der Klasse *PublishMultipleData* als UML-Diagramm.

Die für die ortsfesten Terminals entwickelte Software ist als *PyCharms* Projekt im Anhang 5.8 hinterlegt.

5.3 Mobile Terminals

Zum Zeitpunkt des Abschlusses dieser Arbeit wurden zwei mobile Klima Messstationen (mobile Terminals) gebaut. Die Abbildung 5.15 zeigt ein solches Terminal in seiner Gesamtheit in Betrieb.

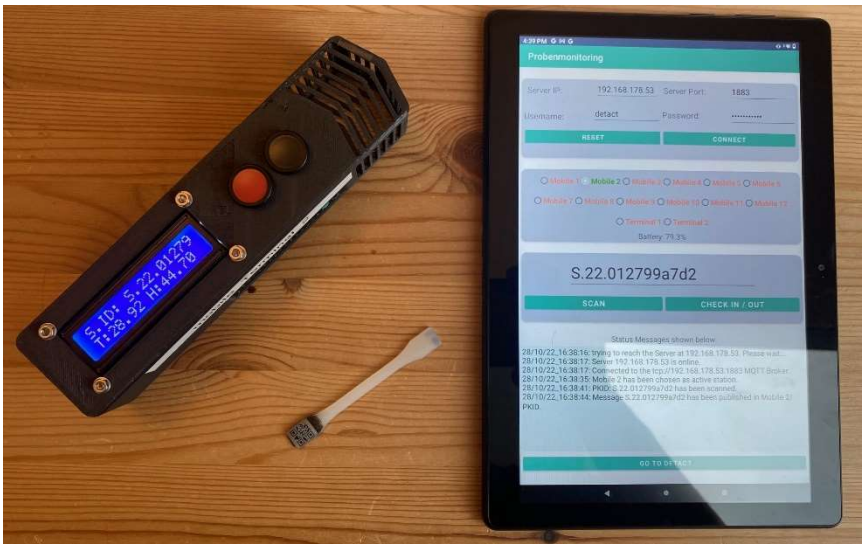


Abbildung 5.15: Mobiles Terminal in Betrieb (rechts), 1BA Probekörper (Mitte) und Android-App zur Steuerung der mobilen Terminals (rechts).

5.3.1 Hardware der mobilen Terminals

Die mobilen Klimamessstationen werden als kompakte Einheiten auf Basis des *Raspberry Pi Zero W v1.1* aufgebaut. Der *Raspberry Pi Zero* ist ein kleiner und ressourcensparender Einplatinencomputer mit einem *ARMv6*-Prozessor mit 1GHz Takt und 512MB Ram. Er ist für den beabsichtigten Anwendungsbereich besonders geeignet, da er kompakt gebaut ist und trotzdem bereits alle nötigen Schnittstellen beinhaltet (insbesondre verfügt er über ein integriertes W-Lan Modul). Technische Daten sind in Anhang A5.9 hinterlegt. Da es sich um die die zweit kleinste Variante der *Raspberry Pi* Serie handelt, hat der *Raspberry Pi Zero* in seiner Grundkonfiguration nur einen geringen Stromverbrauch von etwa 1,8 Watt [95]. Damit ist er als stromsparende Basis für eine akkubetriebene Messstation deutlich besser geeignet als die leistungsstärkeren Modelle des *Raspberry Pi* Line Ups. Die sparsamen *Raspberry Pi Zero* Modelle sind auf der Gegenseite durch ihre geringe Rechenleistung jedoch nur bedingt geeignet QR-Codes kameragestützt auszuwerten. Aus diesem Grund sind die mobilen Terminals nicht als selbstständige Einheiten konzipiert, sondern in der Kombination mit einem

Android basierten Gerät (Smartphone oder Tablet) als System zu verwenden. Mit Hilfe des *Android* Gerätes können Proben gescannt und den mobilen Terminals zugewiesen werden. Für die Kommunikation der Geräte untereinander wird die bereits vorhandene *MQTT*-Infrastruktur benutzt. Abbildung 5.16 verdeutlicht das System der mobilen Terminals.

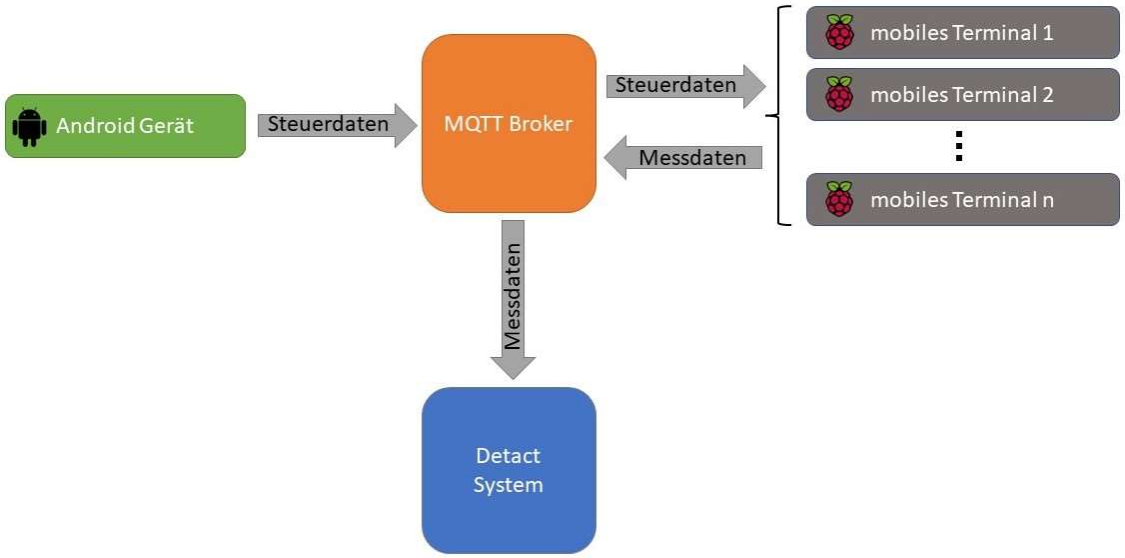


Abbildung 5.16: Aufbau des Systems mobile Terminals [96, 97].

An den *Raspberry Pi* angeschlossen bzw. angebaut sind die folgenden Komponenten:

- **Sensirion SHT31 Temperatur- und Feuchtigkeitssensor:**

  Der Sensor ist baugleich mit dem in den ortsfesten Terminals verbauten Sensor und in Kapitel 5.1.1 bereists ausführlich beschrieben.

- **Waveshare UPS HAT (C) Akkupack:**

  Das *Waveshare UPS HAT (C)* Akkupack sorgt für die Energieversorgung des *Raspberry Pi Zeros,* während er vom Stromnetz getrennt ist. Das Bauteil ist als *Raspberry Pi Zero Hat* (Hutplatte) ausgelegt und lässt sich somit sehr kompakt mit dem *Raspberry Pi* verbauen. Der verbaute Zustand mit einem *Raspberry Pi Zero W* ist in Abbildung 5.17 gezeigt. Das Akkupack ermöglicht den störungsfreien Übergang zwischen Netz- und Akkubetrieb: Die Messstation kann dadurch ohne Unterbrechung der Datenaufnahme vom Stromnetz getrennt, transportiert und wieder an das Stromnetz angeschlossen werden. Die Einheit verfügt über einen 1000mah 3,7V Li-Po Akku. Damit kann der *Raspberry Pi Zero* bis zu 7 Stunden im Standby verweilen. Mit den für die Anwendung des Systems als Klimamessstation

benötigten Anbauteilen ist ein durchgehender Betrieb von 4 Stunden möglich. Die Einheit verfügt über einen separaten Ein/Aus-Schalter und kann über den *I2C* BUS den Batterieladestand sowie aktuelle Verbrauchsdaten übermitteln. Weitere technische Daten sind in Anhang A5.10 hinterlegt.

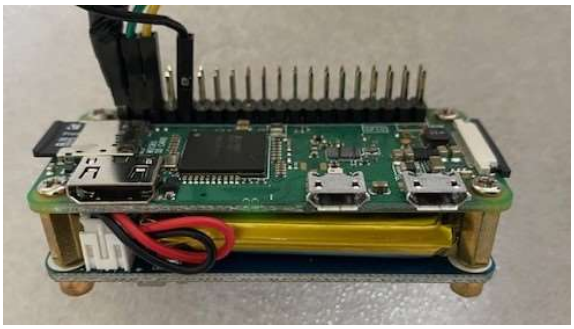


Abbildung 5.17: *Waveshare UPS HAT (C)* verbaut an einem *Raspberry Pi Zero W*.

- **2x16 Zeichen LCD-Display von Joy-It:**

  An den mobilen Terminals selbst ist ein einfaches LCD-Display verbaut. Dieses ist als Ergänzung zur für die zur Steuerung der Messstationen notwendigen Android-App gedacht. Es soll die wesentlichen Informationen für die Bediener*innen auf einen Blick direkt zur Verfügung stellen. Somit zeigt es an, welche Probekörper gerade an dem Terminal angemeldet sind. Auf Knopfduck lässt sich der aktuelle Akkustand des Terminals abrufen. Das Ausgewählte Display kann in zwei Reihen jeweils 16 Zeichen anzeigen, weist eine kompakte Bauform auf und ist mit einem *PCF8574AT* Adapterchip ausgestattet. Somit ist es möglich das Display seriell über den *I2C* BUS anzuschließen, wodurch deutlich weniger *GPIO*-Pins am *Raspberry Pi* belegt werden. Das Display verfügt über eine Hintergrundbeleuchtung. Diese lässt sich über zwei am Display vorhandene Kontakte schalten. Um möglichst wenig Strom zu verbrauchen ist dort ein Taster angebracht (siehe Taster *MCS 18*), sodass das Display nur bei Betätigung des Tasters beleuchtet wird. Weitere technische Daten sind in Anhang A5.11 hinterlegt.

- **Taster MCS 18:**

  An den mobilen Terminals sind jeweils zwei Drucktaster des Typs *MCS 18* angeschlossen. Sie dienen zum Schalten der Beleuchtung des Displays (schwarzer Taster) sowie zum Umschalten der auf dem Display angezeigten Information von „Sample ID“ zu „Akkustand“ (roter Taster). Weitere technische Daten sind in Anhang A5.12 hinterlegt.

- **Micro USB Netzteil Voltcraft DC 5V 2,5A:**

  Für den Netzbetrieb und die Aufladung der mobilen Terminals kommen einfache DC Micro USB Netzteile der Firma *Voltcraft* mit 5,1V und 2,5A zum Einsatz. Weitere technische Daten in Anhang A5.13.

- **Gehäuse des Messsystems:**

  Das Gehäuse ist eine Sonderanfertigung für die mobilen Terminals und im FDM-Verfahren hergestellt. Es besteht aus zwei Teilen. Einem Grundkörper und einem Deckel. In einem abgetrennten Bereich des Gehäuses sind umfassend Schlitze eingebracht, sodass der T/H-Sensor dort im vollen Kontakt mit der Umgebung seht und dennoch geschützt ist. Das Gehäuse besteht aus schwarzem PLA. Sämtliche Konstruktionsdetails sind den angehängten Zeichnungen zu entnehmen (siehe Anhang A5.14 und A5.15). Es ist mit 20% Infill- und Stützelementen gedruckt. Abbildung 5.18 zeigt das leere Gehäuse mit dem Gehäusedeckel.

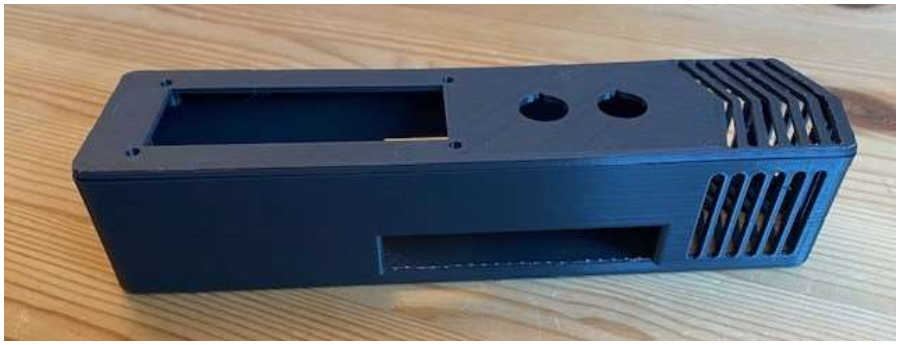


Abbildung 5.18: Leeres Gehäuse der mobilen Terminals aus schwarzem PLA.

- **Android Tablet Lenovo TBX505F:**

  Für die Verwendung der *Probenmonitoring Android* App ist ein *Android* Gerät mit *Android 7* oder höher nötig. Eine große Bildschirmdiagonale ist für die Verwendung der App vorteilhaft, sodass sich insbesondere *Android* Tablets eignen. Das Tablet muss in jedem Fall über eine Kamera verfügen, da sonst ein Einscannen der Probenbezeichnungen als QR-Code nicht möglich ist.

5.3.2 Software der mobilen Terminals

Die Beschreibung der Software innerhalb dieses Kapitels besteht aus zwei Teilen: Der *Raspberry Pi* basierten mobilen Terminals und der *Android*-App *Probenmonitoring*. Auf den *Raspberry Pi Zero W* Einplatinencomputern läuft

Raspberry Pi OS Lite in der Version 11 (*Bullseye*). Die Software für die mobilen Terminals ist in *Python 3.10* unter Verwendung des *PyQT5* Frameworks erstellt und orientiert sich an den unter Kapitel 4.8 beschriebenen Prinzipien. Die *Android*-App ist in *Java*-basiert geschrieben. Sie verwendet nicht das *QT*-Framework. Sie ist damit das einzige Softwareelement, welches sich nicht an allen Prinzipien aus Kapitel 4.8 orientiert.

Beschreibung der Python basierten Software der mobilen Terminals:

Die für die mobilen Terminals entwickelte Software gleicht in großen Teilen der Software für die ortsfesten Terminals. Sie unterscheidet sich jedoch darin, dass es sich nicht um eine Anwendung mit GUI handelt. *PyQT5* basierte Anwendungen können aus diesem Grund nicht mit der in Kapitel 5.1.1 unter Punkt drei beschriebenen *QApplication* ausgeführt werden. Die für den *Signals und Slots*-Mechanismus benötigte *QEventloop* wird abweichend der anderen Anwendungen mit *QCoreApplication* bereitgestellt. Die einzelnen Threads und Elemente werden dann als Workerobjekte in der *QCoreApplication* definiert und ausgeführt. Die Anwendung ist Multithread basiert. Abbildung 5.19 zeigt die Multithreadstruktur der Anwendung. Tabelle 5.2 stellt die Signals und ihre im Main-Thread hervorgerufenen Handlungen dar.

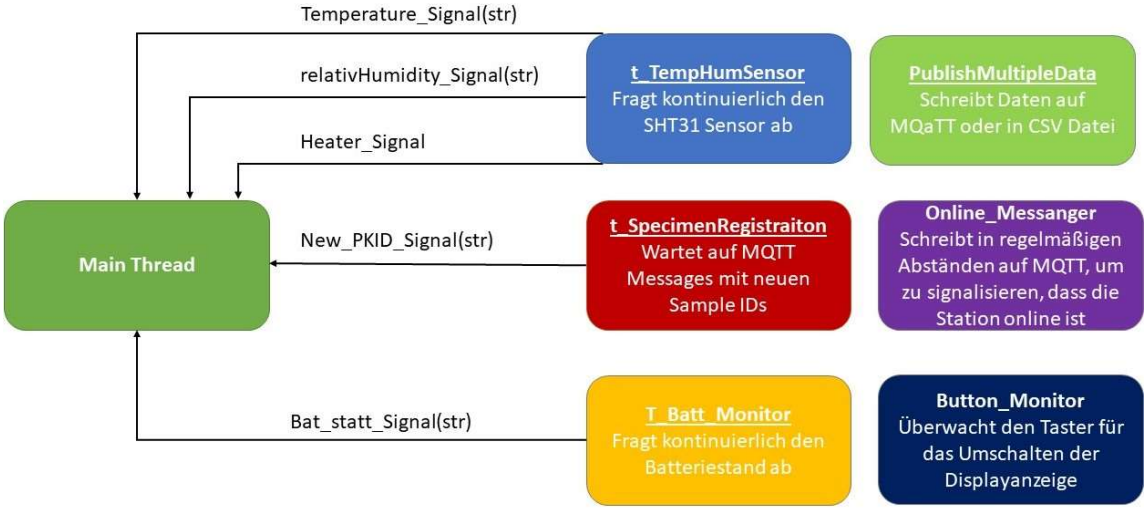


Abbildung 5.19: Darstellung der Multithreadstruktur der mobilen Terminals.

Tabelle 5.2: Darstellung der Signals und ihren im Main-Thread der mobilen Terminals hervorgerufenen Handlungen.

Signal	**Slot**
Temperature_Signal(str)	• Ausgabe des mit dem Signal übermittelten Wertes in der Konsole. • Ergänzung / Aktualisierung des *SpecimenDataFrames* mit dem im Signal übermittelten Wert.
relativ_Humidity_Signal(str)	• Ausgabe des mit dem Signal übermittelten Wertes in der Konsole. • Ergänzung / Aktualisierung des *SpecimenDataFrames* mit dem im Signal übermittelten Wert.
Heater_Signal	• Ausgabe des mit dem Signal übermittelten Wertes in der Konsole.
New_PKID_Signal(str)	• Fügt die mit dem Signal übertragene Sample ID dem *Sample IDs Array* hinzu oder entfernt sie daraus, falls sie darin bereits enthalten ist.
Bat_statt_Signal(str)	• Aktualisierung der Variable *actual_Bat_Capacity* die an den *Button_Monitor* Thread und den *PublishMultipleData* Thread übermittelt wird.

Der Mainthread hat die folgenden Abhängigkeiten:

- TempHumSensor aus t_TempHumSensor
- MQTTPublischer aus t_mqtt_communicator
- BatMonitor aus t_bat_monitor
- MQTTSubscriber aus t_SpecimenRegestration
- Rpi.GPIO [98]
- datetime [72]
- random [80]
- logging [81]
- configparser [65]
- sys [82]
- csv [83]
- json [73]
- Pyqt5.QtCore [84] (Mehrer Elemente)

Nachfolgend werden die einzelnen Worker-Threads beschrieben.

t_TempHumSensor: Beschreibung siehe Kapitel 5.3.2 da es sich um den gleichen Thread handelt.

t_SpecimenRegistration: Das Modul ist zur Abfrage der *MQTT*-Topic „Station_ID“/PKID da. Es erhält von der *Android* App übermittelte *MQTT*-Messages. Das Modul besteht aus der *MQTTPublisher* Klasse des *mqtt_communicator* Moduls, das bereits in Kapitel 5.1 beschrieben ist.

publishMultipleData: Es handelt sich um den gleichen Thread, der bereits bei den ortsfesten Terminals zum Einsatz kommt und in Kapitel 5.3.1 ausführlich beschrieben ist.

t_bat_monitor: Das Modul *t_bat_monitor* ist für die Abfrage des aktuellen Akkustandes des *Waveshare UPS HAT (C)* (siehe Kapitel 5.2.1) über *I2C* BUS zuständig. Der Thread übermittelt den Wert nach Abfrage an den Main-Thread über das *Bat_statt_Signal(str)*. Das Modul besteht aus sechs Klassen welche nachfolgend aufgezählt sind:

1. *BusVoltageRange*, *Gain*, *ADCResolution*, *Mode*, *INA219* (benötigte Klassen für die Busabfrage. Diese stammen von *Waveshare* [99].

2. *BatMonitor*(QThread) Klasse für die Busabfrage als Thread mit Hilfe der in Punkt 1 genannten Klassen. Abbildung 5.20 zeigt den Aufbau der Klasse *BatMonitor*.

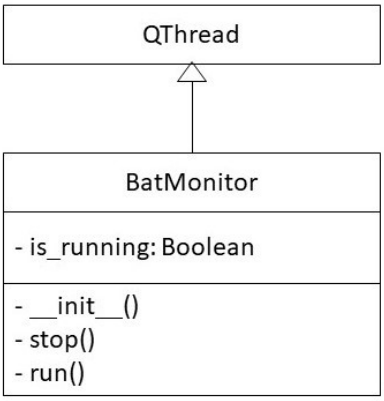


Abbildung 5.20: Aufbau der Klasse *BatMonitor* als UML-Diagramm.

online_Messanger: Der *online_Messenger* Thread übermittelt in einem Intervall von einer Sekunde Daten an den *MQTT-Broker*. Tabelle 5.3 zeigt die *Topics* mit der entsprechenden, übermittelten Information. Mit Hilfe dieser Message kann die *Android*-App erfassen welche mobilen Stationen online und einsatzbereit sind. Darüber hinaus übermittelt die Klasse den aktuellen Akkustand jedes verfügbaren mobilen Terminals, sodass dieser auch aus der App ausgelesen werden kann. Die Klasse *onlineMessenger* ist eine leicht modifizierte Version der *MQTTPublisher* Klasse aus dem *mqtt_communicator* Modul, welches bereits ausführlich in Kapitel 5.1 beschrieben ist. Die Klasse ist um das Erstellen einer „Last Will“ Verfügung (siehe Kapitel 3.2) auf der *MQTT*-Topic „Station_ID“/online ergänzt. Mit dieser Verfügung wird bei Abbruch der Verbindung unter dem Topic „Station_ID“/online die Nachricht „False“ veröffentlicht. Somit kann das nicht mehr erreichbare Terminal in der App als offline angezeigt werden. Die Status Kommunikation wird in Klartext-Messages und nicht als *JSON* vorgenommen.

Tabelle 5.3: Übersicht über die vom *OnlineMessanger* Thread bespielten *MQTT*-Topics und die darin übermittelten Informationen.

Topic	**Übermittelte Informationen**
„Station_ID“/online	Übermittlung eines „True“ Wertes, wenn das mobile Terminal online ist. Bei Verlust der Verbindung wird über die „Last Will“-Funktion des *MQTT*-Protokolls ein „False“ übermittelt.
„Station_ID“/current PKID	Es werden sekündlich die aktuell an dem mobilen Terminal angemeldeten Sample IDs übermittelt.
„Station_ID“/Bat. capacity	Es wird sekündlich der aus dem *T_Bat_Monitor* Thread stammende aktuelle Akkustand übermittelt.

ButtonMonitor: Der Button Monitor Thread fragt kontinuierlich den an den *GPIO*-Pins des *Rasperry Pi Zero* (*GPIO* PIN 24) angeschlossenen roten Taster *MCS 18* (siehe Kapitel 5.2.1) ab. Der Taster wird im Pulldown Modus betrieben. Nach Betätigung des Tasters zeigt das Display für fünf Sekunden den aktuellen Akkustand des mobilen Terminals an.

Die auf den mobilen Terminals laufende Software ist in Anhang A5.8 als *PyCharms* Projekt angefügt.

Beschreibung der Android App:

Die für die Steuerung der mobilen Terminals entwickelte *Android*-App wurde für *Android 11* (API-Level 30) entwickelt. Sie ist abwärtskompatibel bis *Android 7* (API-Level 24). Geschrieben ist die App in *Java 8*. Die *Probenmonitoring* genannte App soll zum Ausgleich der fehlenden Rechenleistung der *Raspberry Pi Zero W* basierten mobilen Terminals dienen. Die App ist in Abbildung 5.21 dargestellt. Die App erfüllt dabei die in Tabelle 5.4 aufgeführten Funktionen.

Abbildung 5.21: Graphische Oberfläche der *Probenmonitoring Android*-App.

Tabelle 5.4: Funktionsübernahme der App gegenübergestellt zur Funktion der ortsfesten Terminals.

Funktion	**Element der App**	**Element des ortsfesten Terminals**
Zugriff auf die Funktionen der Terminals	Android Gerät mit GUI der *MainActivity*	*PyQt5* GUI in Verbindung mit dem *Raspberry Pi* 7“ Touchscreen
Scannen von Sample IDs auf Probekörperetiketten	Scannen mit Hilfe der im Android Gerät verbauten Kamera in der *QRScanActivity*	Scannen mit Hilfe des verbauten *Raspberry Pi* Kameramoduls und dem *t_QrCodeScanner* Moduls
Überblick über die betriebsbereiten Terminals	Anzeige innerhalb der GUI in der *MainActivity*	Keine vergleichbare Funktion
Überblick über die Akkustände der betriebsbereiten mobilen Terminals	Anzeige innerhalb der GUI in der *MainActivity*	Keine vergleichbare Funktion

Die App besteht aus zwei Modulen. Der *MainActivity* und der *QRScannActivity*. Das Modul *MainActivity* beinhaltet den gesamten Funktionsumfang der App, mit Ausnahme der Funktionalität zur Erfassung der QR-Codes, welche in der *QRScanActivity* beinhaltet ist. Das Modul *MainActivity* beinhaltet zwei Klassen, die Klasse *ActivityMain* und die Klasse *Connection_Check_IP*. Das Modul *QRCodeScan* beinhaltet lediglich die Klasse *QRScanActivity*. Die Abbildungen 5.23 und 5.22 veranschaulichen den Aufbau der Klassen *QRScanActivity* und *ActivityMain*.

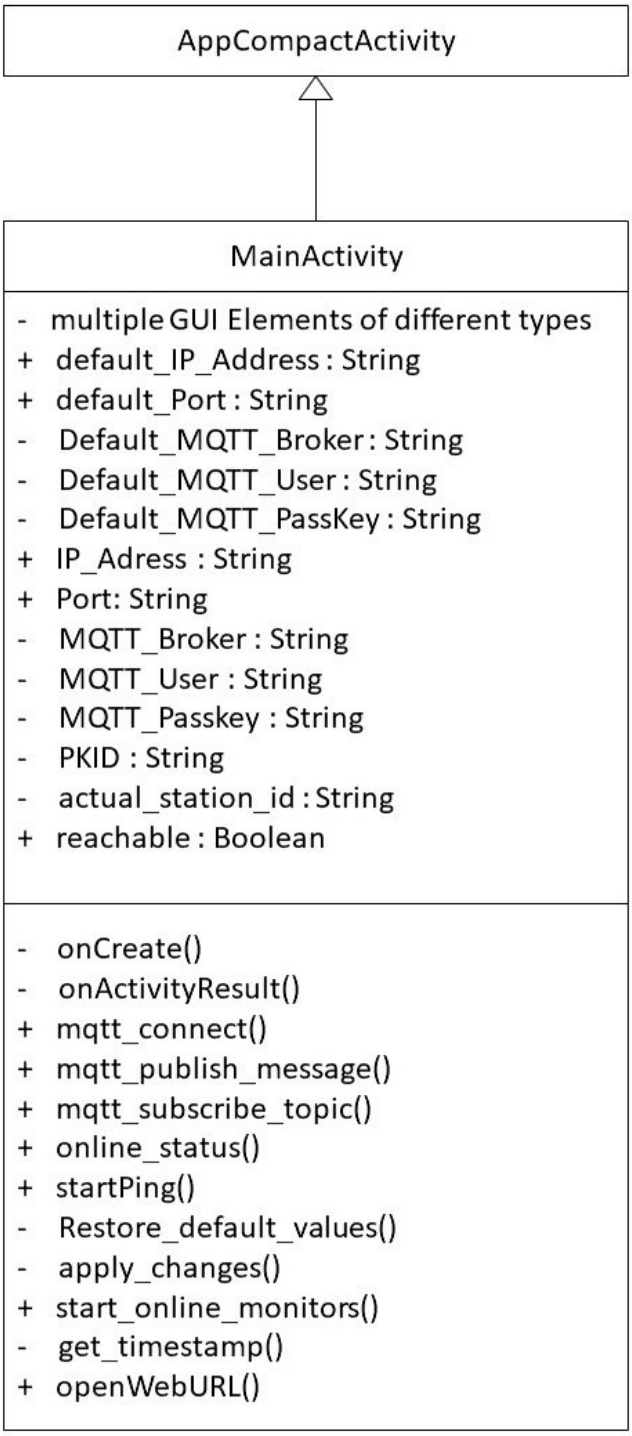


Abbildung 5.22: Aufbau der Klasse *MainActivity* als UML-Diagramm.

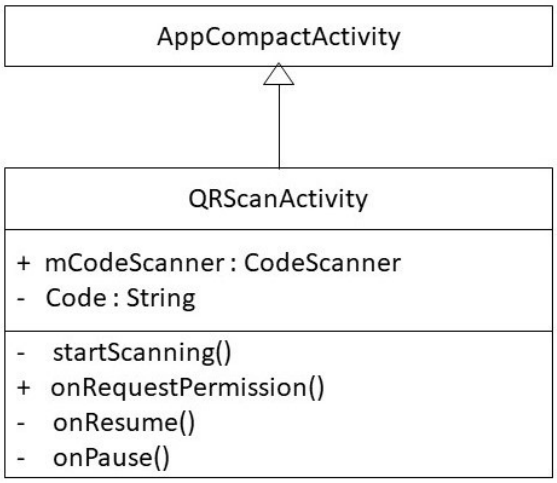


Abbildung 5.23: Aufbau der Klasse *QRScanActivity* als UML-Diagramm.

Der in Kapitel 4.8.1 beschriebene Multithreading Ansatz ist innerhalb der App durch die *Android* eigene *Asynctask* realisiert. Es stellt eine Hilfsklasse zur einfachen Anwendung für die Klassen Thread und Handler dar, welche aus dem *User Interface* (UI) heraus gestartet werden. *Asynctask* ist insbesondere für kurz laufende Netzwerkanwendungen und Aktivitys mit Rückmeldung an das UI vorgesehen.[100] Die Funktion des Scannens der QR-Codes wird durch die *QRScanActivity* realisiert, die über einen *Intent* aus der *MainActivity* angefragt wird. Die *QRScanActivity* übergibt eine gescannte Sample ID zurück an die *MainActivity*. Activitys und Threads (angesteuert durch *Asynctask*) unterscheiden sich darin, dass die Threads gleichzeitig mit der zugehörigen Activity laufen. Bei Activitys ist nur die gerade im Fokus stehende Activity aktiv. Alle anderen werden pausiert [101, 102]. Abbildung 5.24 zeigt den Multithreadaufbau der *Probenmonitoring* App mit *Asynctask*.

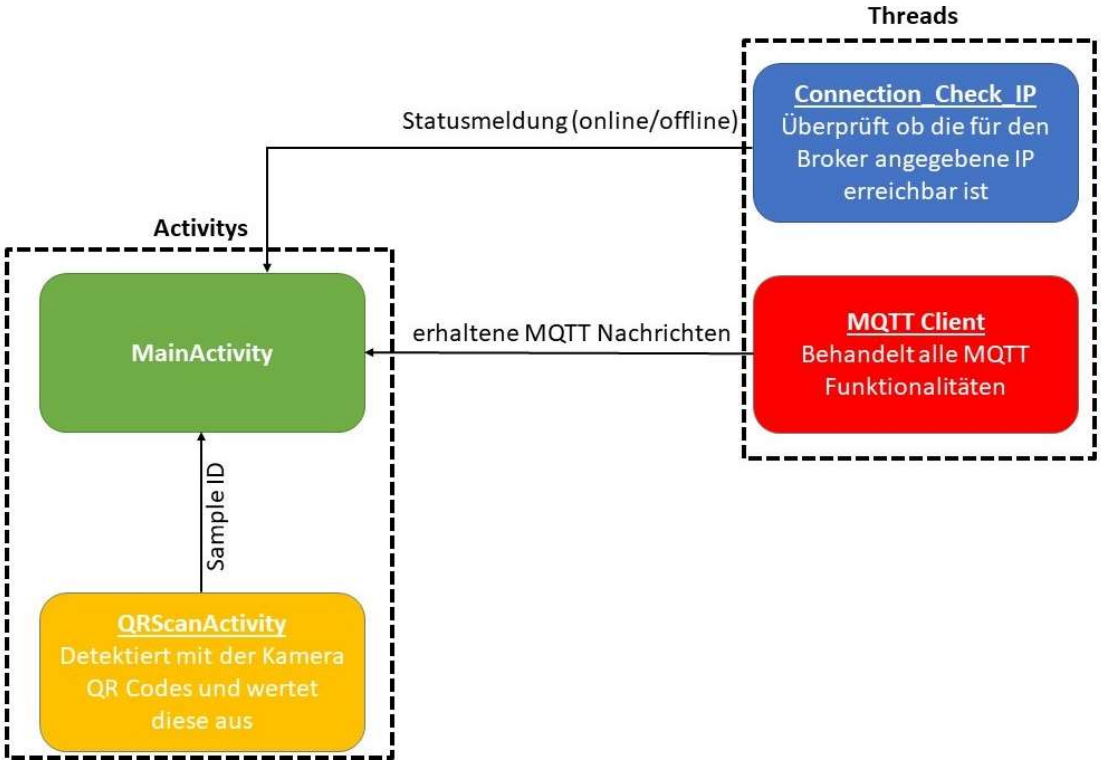


Abbildung 5.24: Multithreading / Activity Übersicht der *Probenmonitoring* App.

Die App hat die folgenden Abhängigkeiten im *Gradle* build:

- androidx.appcompat:appcompat:1.3.1 [103]
- com.google.android.material:material:1.3.0 [104]
- androidx.constraintlayout:constraintlayout:2.0.4 [105]
- androidx.legacy:legacy-support-v4:1.0.0 [106]
- org.eclipse.paho:org.eclipse.paho.android.service:1.0.2 [107]
- androidx.localbroadcastmanager:localbroadcastmanager:1.0.0 [108]
- com.budiyev.android:code-scanner:2.1.0 [109]

Für den Betrieb der App muss sie über die folgenden *Android* Berechtigungen verfügen:

- android.permission.WAKE_LOCK
- android.permission.INTERNET
- android.permission.ACCESS_NETWORK_STATE
- android.permission.READ_PHONE_STATE
- android.permission.CMAERA

Die wichtigsten Methoden der *ActivityMain* sind nachfolgend beschrieben:

onCreate(): Die *onCreate* Methode wird bei der Erstellung des Objektes ausgeführt und dient als Initialisierungsmethode. Innerhalb dieser Methode werden die Elemente aus der GUI mit den entsprechenden Methoden der *MainActivity* verknüpft (genauere Beschreibung der GUI Funktionen in Anhang A5.16). Innerhalb dieser Methode werden auch Standardparameter für die *MQTT*-Zugangsdaten gesetzt. Anschließend wird die Erreichbarkeit des *MQTT-Brokers* überprüft und die Online Monitore als *MQTT*-Subscriber gestartet.

onActivityResult(): innerhalb dieser Methode werden die aufgenommenen Sample IDs aus der *QRScanActivity* in die *MainActivity* übergeben.

mqtt_connect(): Die *mqtt_connect()* Methode erstellt ein *MQTT*-Client Objekt und konfiguriert es entsprechend mit den mitgegebenen Zugangsdaten. Es wird innerhalb dieser Methode versucht eine Verbindung aufzubauen und das Ergebnis des Versuches des Verbindungsaufbaus innerhalb der GUI kommuniziert.

mqtt_publish_message(): Veröffentlicht mit Hilfe des in *mqtt_connect()* erstellten Client Objekts eine Nachricht auf dem *MQTT-Broker*. Topic und Nachricht werden der Methode als Argumente mitgegeben.

online_status(): Die Methode fragt mit Hilfe des in *mqtt_connect()* erstellten Client Objekts die drei Topics „Station_ID“/online, „Station_ID“/current PKID, „Station ID“/Bat. Capacity kontinuierlich ab. Die Methode zeigt die übermittelten Informationen entsprechend innerhalb der GUI an.

startPing(): Initalisiert ein Objekt der *Connection_Check_IP* Klasse und startet die asynchrone Ping Methode. Die Methode wird von der Methode *onCreate* aufgerufen.

start_online_monitors(): Für den Fall, dass der *Broker* erreichbar ist, ruft die Methode für jedes vorhandene mobile Terminal die Methode *online_status()* auf. Die Methode wird von der *onCreate* Methode aufgerufen.

Die wichtigsten Methoden der *QRScanActivity* sind nachfolgend beschrieben:

onCreate(): Die *onCreate* Methode wird bei der Erstellung des Objektes ausgeführt und dient als Initialisierungs Methode. Innerhalb dieser Methode werden die benötigten Berechtigungen abgefragt.

startScanning(): Die Methode initialisiert ein *CodeScanner* Objekt und richtet ein Callback ein, welches nach detektieren eines Codes im Kamera stream den Code mit einem Intent an die *MainActivity* zurück gibt.

Die entwickelte Software ist als *Android Studio* Projekt im Anhang A5.17 zu finden.

5.4 Zugprüfmaschinen Zwick/Roell ZwickiLine Z2.5

Am Institut ist eine Universalprüfmaschine der Firma *Zwick/Roell* (Deutschland) des Typs *ZwickiLine Z2.5* vorhanden. Die Maschine wird mit Hilfe eines an sie angeschlossenen Messrechners bedient und mit der Software *TestExpert III* betrieben [110]. Mit dieser Software ist es möglich, vollständige Versuchsreihen durchzuführen und die dabei entstandenen Daten zu analysieren, zu ordnen und in unterschiedlichen Formaten abzulegen. Die Software läuft auf dem an der Maschine angeschlossen *Windows* 10 Rechner. Abbildung 5.25 zeigt eine solche Maschine mit der *TestExpert III* Software.

Abbildung 5.25: *ZwickiLine* Prüfmaschine mit sichtbarer *TestExpert III* Software auf dem Messrechner [111].

5.4.1 Hardware der Universalprüfmaschine

Die *ZwickiLine* Prüfmaschinen sind speziell für die Prüfung von Kunstoffen, Elastomeren und Metallen entwickelte Materialprüfmaschinen. Sie sind mit bis zu einer maximalen Prüfkraft von 5kN verfügbar [112]. Die am Institut befindliche Maschine hat eine maximale Prüfkraft von 2,5kN. Die Maschine kann durch unterschiedliche Anbauteile für unterschiedliche Materialprüfverfahren genutzt werden. Insbesondere sollen hier der einachsige Zugversuch, wie auch der

Dreipunktbiegeversuch erwähnt sein. Die Maschine ist mit der *TestControl II* Mess- und Regelelektronik von *Zwick* ausgestattet [112]. Weitere technische Daten zur Maschine sind dem Datenblatt in Anhang A5.18 zu entnehmen. Eine direkte Verbindung der Maschine mit dem *Detact* System ist nur durch eine Zukaufleistung des Herstellers möglich.

Die mit Sample ID versehenen Probekörper sollen an der Maschine selbst mit ihrer Sample ID in der *TestExpert III* Software eingepflegt werden. Dazu ist ein USB-Tischscanner der Marke *Kaptur* des Typs *KP1401* an dem Messrechner installiert. Der Scanner kann 1D und 2D Barcodes erfassen [113] (Datenblatt des *Kaptur* Scanners siehe Anhang A5.19). Er arbeitet durch Identifikation als Tastatur am Hostgerät. Somit ist es möglich durch Auswahl des Feldes für die Probekörperbenennung innerhalb der *TestExpert III*-Software das Feld durch Scannen des Codes vollständig auszufüllen. Das Scannen der Probekörperetiketten ist der einzige, zusätzliche Schritt im Arbeitsablauf für die Maschinenbediener*innen und ersetzt den händischen Eintrag von Probennamen in der Software. Ansonsten fällt kein zusätzlicher Aufwand für den Import der Daten in das *Detact* System an.

5.4.2 Prüfmaschinensoftware und Detact Import Tool

Die *TestExpert III* Software bietet eine vollumfängliche Kontrolle über die Maschine. Nach Abschluss eines Versuches oder einer Versuchsreihe bietet die *TestExpert III* Software die Möglichkeit die aufgenommenen Daten zu exportieren. Dazu stehen unterschiedliche Exportformate bereit. Diese sind unter andrem CSV, Ascii, PDF oder *Excel*-Tabellen. Die in *Excel* Dateien vorliegenden Daten zu den Versuchsreihen werden in einem stets gleichen Verzeichnis abgelegt. Die im Rahmen dieser Arbeit für die Prüfmaschine entwickelte Software zur Einpflegung der Versuchsdaten in das *Detact* System beruht auf der Verwendung der eben beschriebenen *Excel* Exporten. Die Datentransfersoftware läuft auf dem Messrechner der Prüfmaschine. Sie wird automatisch nach dem Hochfahren des Rechners gestartet und überwacht das Verzeichnis, in welches die Versuchsdaten nach Abschluss einer Versuchsreihe exportiert werden, extrahiert die aus dem *Excel* Export benötigten Daten und sendet sie über *MQTT* an das *Detact* System. Dieses Vorgehen bietet den Vorteil, dass das ursprüngliche Prüfsystem aus Prüfmaschine und Messrechner nicht beeinflusst oder manipuliert wird. Es hat den Nachteil, dass die Daten nicht simultan zum Versuch auf das *Detact* System übertragen werden. Die Daten werden also immer erst nach Abschluss des Versuches an das *Detact* System übermittelt. Die graphische Oberfläche des Import Tools ist in Abbildung 5.26 abgebildet. Der Prozess der Datengewinnung

für das *Detact* System ist in Abbildung 5.27 dargestellt. Nach dem hier beschriebenen Vorgehen lassen sich auch weitere Prüfmaschinen, welche in Dateien exportieren an das *Detact* System ankoppeln.

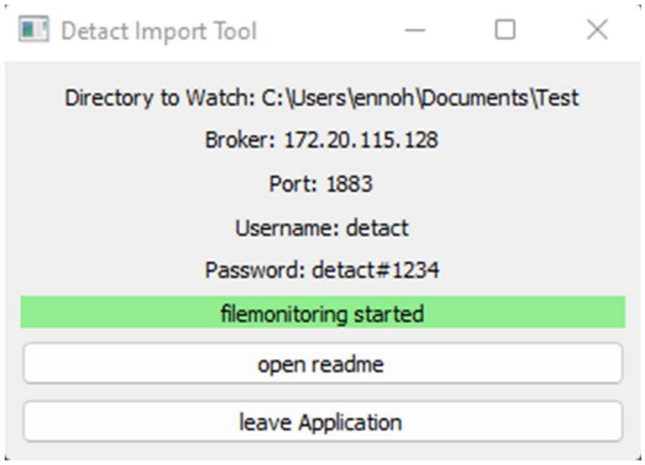


Abbildung 5.26: Graphische Oberfläche des *Detact* Import Tools.

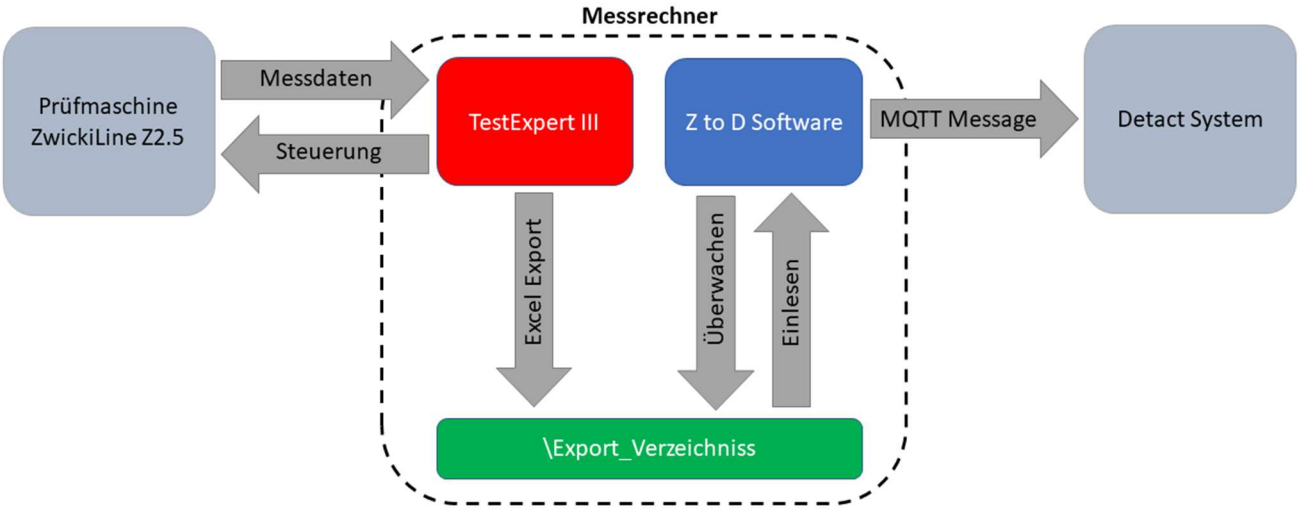


Abbildung 5.27: Darstellung der Datengewinnung für das *Detact* System an der *ZwickiLine* Prüfmaschine.

Die *Zwick* to *Detact* Import-Software (*Z to D Import Tool*) wurde in *Python 3.10* unter Verwendung des *PyQt5* Frameworks geschrieben. Sie orientiert sich an dem in Kapitel 4.8.1 beschrieben Multithreading Ansatz. Abbildung 5.28 zeigt den Aufbau der Threads und Signals der *Zwick* to *Detact* Import Software.

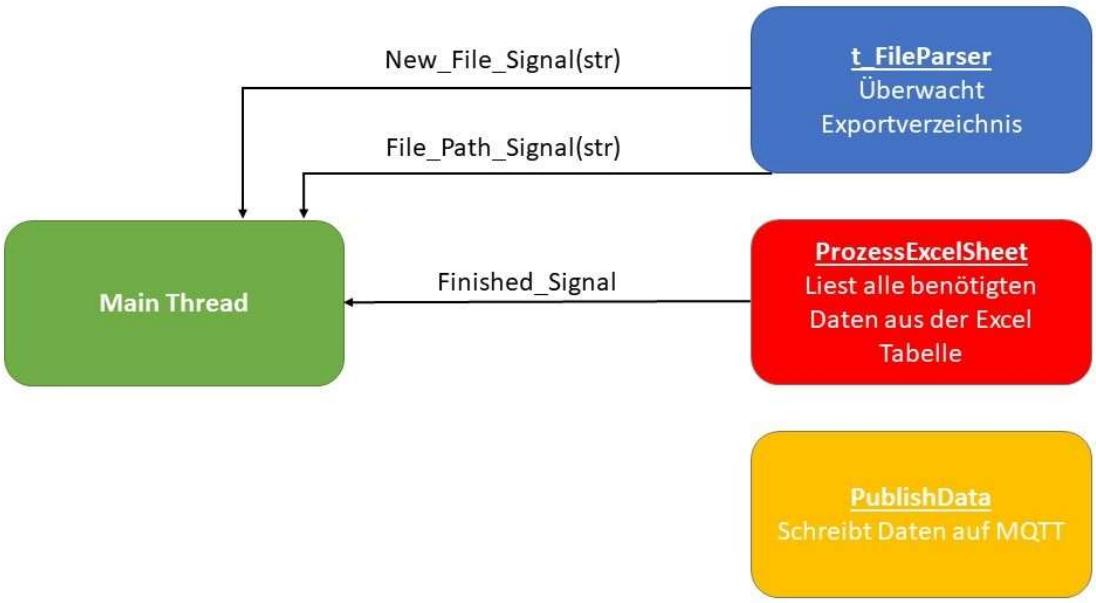


Abbildung 5.28: Threadstruktur der *Zwick to Detact* Import Software.

Main-Thread: Neben den in Kapitel 5.1.1 genannten Funktionalitäten stellt der Main-Thread die folgenden Funktionen bereit:

- Starten der Worker-Threads.
- Initialisierung der GUI. (siehe Abbildung 5.26)
- Slots zu den ankommenden Signals aus den Worker-Threads. (Siehe Tabelle 5.5)

Tabelle 5.5: Darstellung der Signals und ihren im Main-Thread hervorgerufenen Handlungen.

Signal	**Slot**
New_File_Signal(str)	Loggen der Detektion einer neuen Datei im Exportverzeichnis. Das Signal übermittelt den Namen der Datei.
File_Path_Signal(str)	Wenn die detektierte Datei vollständig abgelegt ist, wird der Dateipfad an den Main-Thread übermittelt und der *ProzessExcelThread* gestartet.
Finished_Signal	Wenn der *ProzessExcelThread* abgeschlossen ist, wird dem Benutzer / der Benutzerin angezeigt, dass der Import abgeschlossen ist.

Der Main-Thread hat die folgenden Abhängigkeiten:

- MQTTPublischer aus t_mqtt_communicator
- FileParser aus t_fileparser
- datetime [72]
- logging [81]
- configparser [65]
- sys [82]
- os [114]
- pandas [115]
- json [73]
- Pyqt5.QtCore [84] (Mehrere Elemente)
- Pyqt5.Qwidgets [86] (Mehrere Elemente)

Die durch den Main-Thread vorgenommene Konfiguration mit Werten aus der Initialisierungsdatei config.ini wird in Anhang A5.20 geschildert.

t_fileparser: Das *t_fileparser* Modul beinhaltet die Klasse *FileParser*, welche von der *QThread* Klasse erbt. Mit Hilfe dieser Klasse kann ein Verzeichnis auf Veränderungen in Form von neuen Dateien in dem Verzeichnis überwacht werden. Die Klasse übermittelt zwei Signals: Zunächst das *new_file_Signal(str)* mit welchem signalisiert wird, dass ein neues Signal erkannt wurde, sowie das *File_Path_Signal*. Letzteres wird emittiert, wenn die erkannte Datei vollständig in dem Verzeichnis abgelegt wurde und der vollständige Dateipfad mit dem Signal übermittelt ist. Die Klasse bedient sich der *QFilesystemWatcher* Klasse und reagiert auf das *DirectoryChanged*-Signal dieser Klasse. Abbildung 5.29 zeigt den Aufbau der *FileParser* Klasse.

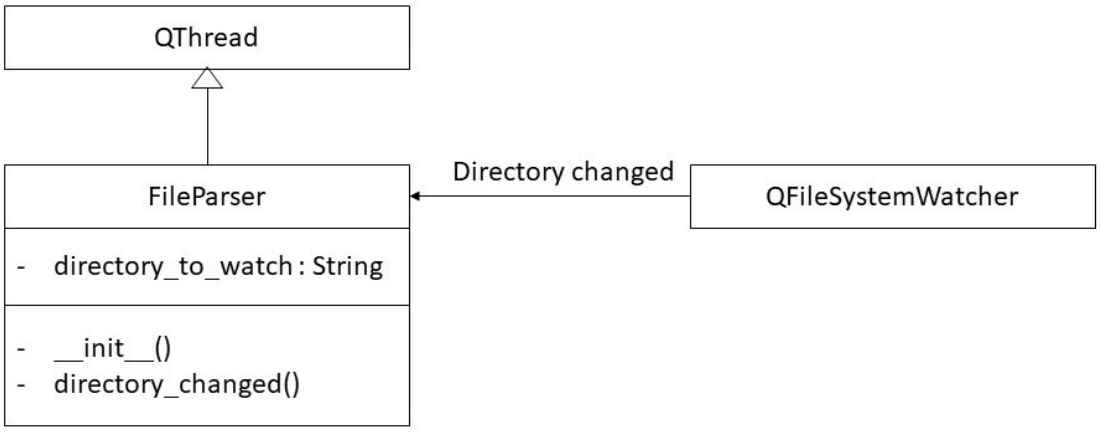


Abbildung 5.29: Aufbau der Klasse *FileParser* als UML-Diagramm.

Die Klasse *FileParser* hat die folgenden Abhängigkeiten:

- QThread [67], pyqtsignal [87], pyqtslot [88], QFileSystemWatcher [116]
- time [89]
- logging [81]
- pandas [115]

PublishData: Der *PublishData* Thread funktioniert analog zu dem im Rahmen der *Probenmonitoring* Software beschriebenen *PublishMultipleData* Threads. Er unterscheidet sich jedoch darin, dass er nur für eine einzelne Sample ID agiert und keine CSV-Dateien anlegt. Der Thread veröffentlicht die aktuell im *SpecimenDataFrame* hinterlegten Daten als *JSON*-String in einer *MQTT*-Message. Dieser benutzt die *MQTTPublisher* Klasse aus dem *mqtt_communicator* Modul. Abbildung 5.30 zeigt den Aufbau der *PublishData* Klasse.

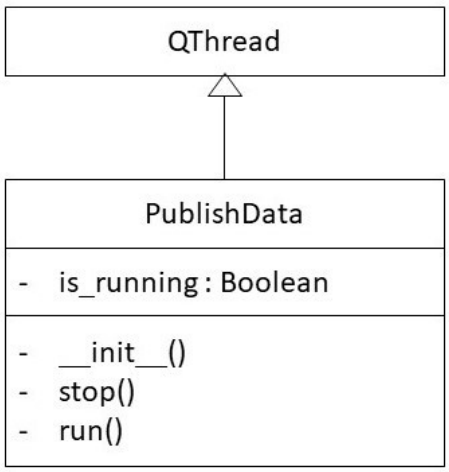


Abbildung 5.30: Darstellung des Aufbaus der *PublishData* Klasse als UML-Diagramm.

ProzessExcelSheet: Der *ProzessExcelSheet* erfüllt die Hauptaufgabe *des Detact Import Tools.* Der Thread liest mit Hilfe von *Pandas* [115] die Daten aus der von der *TestExpert III* exportierten *Excel* Datei aus. Die Hauptmethode des Threads heißt *process_excel_sheet()* Diese *Excel* Datei hat mehrere Arbeitsblätter, auf denen unterschiedliche Datenarten abgelegt sind. Der Aufbau der einzelnen *Excel* Arbeitsblätter ist wie folgt: Auf dem ersten Arbeitsblatt sind Angaben zu den Prüfparametern hinterlegt. Auf dem zweiten Arbeitsblatt werden die ermittelten, mechanischen Kennwerte der Proben der Testreihe nacheinander aufgelistet. Auf Arbeitsblatt drei werden statistische Daten gelistet. Diese statistischen Daten werden beim Import nicht berücksichtigt. Anschließend folgt für jede Probe der Versuchsreihe je ein Arbeitsblatt mit Messwerten. Die Arbeitsblätter können unterschiedliche Messgrößen enthalten. Je Messgröße gibt es einen „Channel“ der als eine Spalte in der *Excel* Tabelle repräsentiert wird. Es können beliebig viele Channels eingelesen werden. Der Channel „Prüfzeit“ muss zu jeder Probe vorliegen. Nur so ist es möglich die Messergebnisse mit, in Relation zueinander, richtigen Zeitstempeln zu versehen. Die Methode wertet pro Probe als erstes die zeitbezogenen Daten von den entsprechenden Arbeitsblättern der *Excel* Datei mit allen Channels aus. Es wird pro Zeitpunkt in der Exportdatei ein *SpecimenDataFrame* erstellt und auf *MQTT* als *JSON* veröffentlicht. Anschließend werden die stationären Daten in den *SpecimenDataFrame* eingelesen und auf *MQTT* als *JSON* veröffentlicht.

Zur Erhöhung der Benutzerfreundlichkeit ist für die Software *Detact Import Tool* mit *Inno Setup* ein Installationspaket erstellt worden, welches die Software mit allen benötigten Abhängigkeiten auf *Windows 7* Rechnern (oder höher) installiert.

Somit kann die Software auch von nicht mit der Programmierung vertrauten Personen aufgespielt werden. Das *Inno Setup* Skript befindet sich in Anhang A5.21. Die Software ist als *PyCharm* Projekt in Anhang A5.22 hinterlegt.

5.5 ID Vergabe Software

Die ID Vergabe Software ist das nötige Werkzeug zur Erzeugung der in Kapitel 4.2 beschrieben Probekörperbezeichnung. Diese wird im Kontext dieser Arbeit Sample ID genannt. Dieses Kapitel gibt einen Überblick über den Aufbau der dafür nötigen Hard- und Software.

5.5.1 Hardwareanforderungen der ID Vergabe Software

Die ID-Vergabesoftware ist auf allen aktuellen *Windows* Systemen (*Windows 7* und aufwärts) lauffähig. Für den vollständigen Betrieb ist ein beliebiger mit dem Hostsystem verbundener Drucker oder ein anderes Beschriftungsgerät (z.B. Laserbeschrifter) nötig. Für das reine Erstellen von Sample IDs mit der Software ist ein Beschriftungsgerät aber nicht essenziell.

5.5.2 Beschreibung der ID Vergabe Software

Die *ID-Vergabesoftware* wurde in *Python 3.10* unter Verwendung des *PyQt5* Frameworks geschrieben. Sie orientiert sich an den in Kapitel 4.8.1 beschrieben Multithreading Ansatz. Abbildung 5.31 zeigt die graphische Oberfläche der *ID-Vergabesoftware*. Abbildung 5.32 zeigt den Aufbau der Threads *ID-Vergabesoftware*.

Abbildung 5.31: Graphische Oberfläche der ID-Vergabesoftware.

Abbildung 5.32: Threadstruktur der ID-Vergabesoftware.

Für die Vergabesoftware werden keine spezifischen Signals definiert. Die meisten Aufgaben sind auf Grund ihrer kurzzeitigen Arbeitsdauer im Main-Thread definiert und werden auch von ihm ausgeführt. Lediglich die Kommunikation über *MQTT* findet in einen gesonderten Worker-Thread statt. Für diesen werden jedoch keine weiteren Informationssignals benötigt.

Main-Thread: Neben den in Kapitel 5.1.1 genannten Funktionalitäten stellt der Main-Thread die folgenden Funktionen bereit:

- Starten des Worker-Threads.
- Initialisierung der GUI. Diese ist in Abbildung 5.30 dargestellt.
- Bereitstellen und Ausführen der wesentlichen Arbeitsmethoden für den Betrieb der Software. Diese sind im Folgenden beschrieben.
    - **timestamp_posix():** Die Methode erstellt den in Kapitel 4.2 „Aufbau der Probekörperkennzeichnung“ beschriebenen, modifizierten *Posix*-Zeitstempel. Der erstellte Zeitstempel ist der wesentliche Baustein des Sample ID Bodys und stellt das nicht reproduzierbare Merkmal der Sample ID dar. Abbildung 5.33 zeigt den Aufbau der Methode als Flussdiagramm. Die Methode gibt den modifizierten *Posix*-Zeitstempel als String zurück.

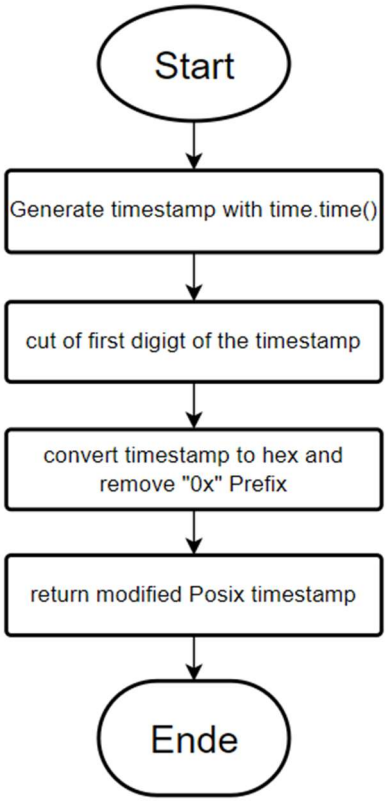


Abbildung 5.33: Aufbau der Methode *timstamp_posix()* als Flussdiagramm.

- **add_ID_to_Image():** Die Methode fügt eine erstellte Sample ID unter einen QR Code Label hinzu, welches die selbe Sample ID repräsentiert. Das Hinzufügen der Sample ID in Klartext wird mit dem Modul *Pillow* [117] realisiert.
- **generate_id():** Die Methode stellt die wesentlichen Bestandteile der Sample ID zum aktuellen Zeitpunkt zusammen. Sie fragt die Zeitstempel an und setzt die ID mit Präfix 1 und Präfix 2 der ID der ID-Vergabestation sowie dem modifizierten *Posix* Zeitstempel aus *timestamp_posix()* zusammen. Zu der erzeugten Sample ID wird innerhalb der Methode mit den Modulen *qrcode* [52] und *Pillow* [117] ein QR Code mit 30% Fehlerkorrektur als *Portable Network Graphic* (PNG) erstellt. Anschließend wird mit der Methode *add_ID_to_Image()* die Sample ID als Klartext unter dem QR-Code und die Sample ID dem *SpecimenDataFrame* hinzugefügt. Im letzten Schritt wird der *SpecimenDataFrame* mit dem Thread *PublishData* über *MQTT* an das *Detact* System übermittelt.

Der Main-Thread hat die folgenden Abhängigkeiten:

- MQTTPublisher aus t_mqtt_communicator
- datetime [72]
- logging [81]
- time [89]
- PIL [117]
- qrcode
- configparser [65]
- sys [82]
- os [114]
- pandas [115]
- json [73]
- Pyqt5.QtCore [84] (Mehrer Elemente)
- Pyqt5.Qwidgets [86] (Mehre Elemente)
- Pyqt5.QPrintSupport [118]
- Pyqt5.QtGui [85]

PublishData: Der eingesetzte Thread für die *MQTT*-Kommunikation ist identisch mit dem bereits in Kapitel 5.3.1 beschriebenen Thread.

Die ID Vergabe Software ist so konzipiert, dass sie auf möglichst vielen Rechnern betrieben werden kann. Daraus resultiert die Möglichkeit an vielen Orten gleichzeitig Sample IDs erstellen zu können. Damit auch die Einrichtung einer neuen ID Vergabe Station unkompliziert ermöglicht wird, ähnlich zur *Detact Import Software,* wurde ein Installationspaket mit allen nötigen Abhängigkeiten der Software erstellt. Das *Inno Setup* Skript dazu befindet sich in Anhang A5.23. Die Software ist als *PyCharm* Projekt in Anhang A5.24 hinterlegt. Eine Bedienungsanleitung für die ID Vergabe Software ist in Anhang 5.25 abgelegt.

6 Experimentelle Versuchsdurchführung und FE-Berechnung

Zur Validierung des in Kapitel 2.2 beschriebenen *Abaqus* Simulationsmodells für die Sorptionsrechnung von PA6 Probekörpern wurden, mit Hilfe der Monitoringumgebung im Rahmen einer Versuchsreihe, PA6 Probekörper nachverfolgt. Bei den erhobenen Daten handelt es sich um Klimadaten und mechanische Daten. Innerhalb dieses Kapitels wird zunächst in Kapitel 6.1 beschrieben wie auf die an das *Detact* System übermittelten Daten abgerufen werden können. Anschließend werden in Kapitel 3.2 die durchgeführten Versuche in ihrem Aufbau und ihrer Durchführung geschildert. Kapitel 3.3 befasst sich mit der Gegenüberstellung der experimentell ermittelten und der numerisch berechneten Daten. In Kapitel 3.4 wird eine Bewertung der Ergebnisse vorgenommen.

6.1 Datenbereitstellung durch das Detact System

Die auf unterschiedlichen Kanälen im *Detact* System zusammengetragenen Probendaten können auf unterschiedliche Arten bereitgestellt werden. Neben der direkten Betrachtung der Parameterdaten im *Detact* System selbst sind die folgend beschriebenen Wege zur Bereitstellung der gesammelten Daten möglich:

- **Datenbereitstellung durch Export einer CSV-Datei:** Innerhalb des *Detact* Systems ist es möglich im *Fertigungslose* Bereich des Systems einen *.csv* Export ausgewählter Daten zu erstellen und gebündelt herunterzuladen.
- **Datenbereitstellung über *REST*-Schnittstelle:** Zum Zeitpunkt der Bearbeitung dieser Arbeit ist die *REST*-Schnittstelle des *Detact* Systems noch nicht ausreichend mit dem Simulationssystem verknüpft. Die Dokumentation der Schnittstelle befindet sich im Anhang unter A6.1.

6.2 Versuchsaufbau und Durchführung

Für die Durchführung der Versuche zur Validierung der numerischen Berechnung der Massendiffusionsanalyse in Kombination mit einer nachgeschalteten statischen Spannungsanalyse werden mehrere 1BA Kunststoff Mehrzweckprobekörper aus PA6 verwendet [46]. Die Probekörper wurden zuvor auf einer Spritzgießmaschine des Typs *Allrounder 470A* von *Arburg* (Deutschland) in einer Fertigungscharge spritzgegossen und die entsprechenden, zum jeweiligen Schuss der Maschine zugehörigen Sensordaten wurden im *Detact* System für jeden hergestellten Probekörper eindeutig zugeordnet und hinterlegt. Damit von einem vollständig trockenen Ausgangszustand vor der Konditionierung ausgegangen werden kann und um eventuell ablaufende Kristallisationsprozesse bereits weitestgehend vor dem Start der Konditionierung ablaufen zu lassen, werden alle hergestellten Probekörper für 100 h bei 80 °C im Vakuumofen getempert. Anschließend werden die Probekörper für die durchzuführenden Versuchsreihen unterschiedlich konditioniert, wobei die Klimadaten (Temperatur und relative Luftfeuchtigkeit) der Proben fortlaufend erfasst werden. Nach einem Zeitintervall von zwei Wochen werden die Proben an der *Zwick/Roell* Universalprüfmaschine (siehe Kapitel 5.4) im einachsigen Zugversuch geprüft. Die Probekörper haben die nachfolgend beschriebenen Konditionierungshistorien:

- **Probensatz 1:** Zehn Probekörper werden zur Sorption bei ungeregeltem Laborklima ausgelagert.
- **Probensatz 2:** Zehn Probekörper werden in einem Klimaofen gelagert. Im Klimaofen werden unterschiedliche relative Luftfeuchtigkeiten bei einer konstanten Temperatur von 23°C eingestellt. Durch die vorgegebenen, unterschiedlichen relativen Luftfeuchtigkeiten soll ein Klima eingestellt werden, bei dem sowohl Sorptions- als auch Desorptionsprozesse stattfinden. Das verwendete Klimaprofil wird in Abbildung 6.2 dargestellt.
- **Probensatz 3:** Zehn Probekörper werden vollständig gesättigt (Konditionierung im Wasserbad bei 80°C für 30 Stunden anschließend Konditionierung bei 23°C im Wasserbad für 9 Tage). Die Proben werden gesättigt zur Rücktrocknung im Laborklima (siehe auch Probensatz 1) ausgelagert.

Vor dem Start der Versuche werden alle Proben mit einer Sample ID und einem QR-Code Etikett ausgestattet, sodass sie über den Zeitraum der Versuche vollständig identifizierbar sind. Die QR-Code Etiketten werden mit der in Kapitel

5.5 beschriebenen Software erstellt und an das *Detact* System übermittelt.

Während der Konditionierung werden die Umgebungsklimadaten der Proben kontinuierlich überwacht. Zu diesem Zweck werden die Proben jeweils an einem ortsfesten Klimaterminal (siehe Kapitel 5.2) angemeldet (Siehe Abbildung 6.4). Die Messpunkte der Terminals liegen im Intervall von 6 Sekunden (= 0,17 Hz). Die aufgenommenen Daten werden im *Detact* System hinterlegt und zusätzlich parallel auf den ortsfesten Klimaterminals selbst mit Hilfe des CSV Logger Moduls in *.csv* Dateien gespeichert. Dies dient bei diesen ersten Versuchsreihen der Erzeugung einer Redundanz der Daten aus Sicherheitsgründen. Begleitend dazu werden die Proben an Werktagen täglich gewogen. Über das Gewicht kann nach Formel 6.1 die Wasseraufnahme der Proben bei bekanntem Trockengewicht ermittelt werden. Das Gewicht wird zum Zeitpunkt der Durchführung der Versuchsreihen händisch erfasst. Abbildung 6.1 zeigt die aufgenommenen Gewichte der Versuchsreihen. Die prozentuale, integrale Massenzunahme pro Versuchsreihe ist als arithmetisches Mittel der Einzelgewichte der Probekörper einer Versuchsreihe aufgetragen.

$$M(t) = \left(\frac{m}{m_0} - 1\right) * 100 \tag{6.1}$$

mit	$M(t)$	Integrale Masse
	m	Aktuelle Masse
	m_0	Ausgangsmasse

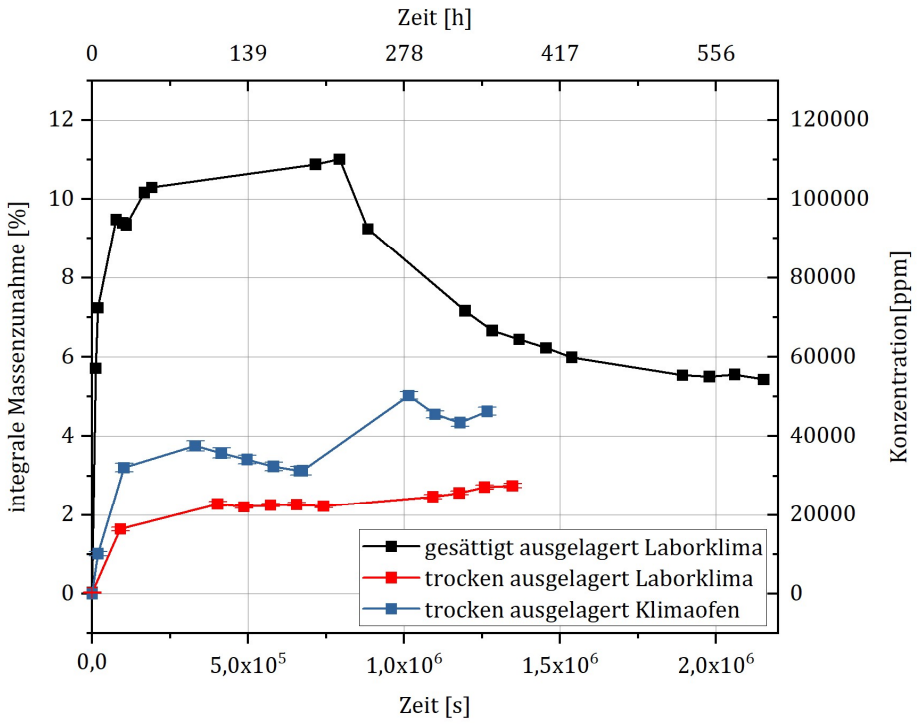


Abbildung 6.1: Durch wiegen ermittelte Gewichte der Probekörper im arithmetischen Mittel für die Probensätze eins, zwei und drei. Die gesättigt ausgelagerten Proben wurden bereits im Wasserbad regelmäßig gewogen, wobei der Wechsel vom Wasserbad bei 80 °C in das Wasserbad bei 23 °C bei 2,5 x 10^6 s stattfand und der Wechsel in das klimaüberwachte Labor bei ca. 7,5 x 10^6 s durchgeführt wurde.

Die Proben der Probensätze eins & drei sind im Labor an einem ortsfesten Klimaterminal (siehe Kapitel 5.2) zur Überwachung angemeldet. Die Proben werden dabei auf einer Tischoberfläche gelagert. In der unmittelbaren Umgebung befanden sich keine elektrischen Geräte, welche durch Abwärme die Klimadatenaufnahme verfälschen könnten. Die aufgenommenen Klimadaten sind den Abbildungen 6.2 und 6.3 zu entnehmen. Der Probensatz zwei wurde in einem Klimaschrank der Firma *Memmert* (Deutschland) *Typ CTC 256* einem sich definiert verändernden Klima ausgesetzt. Der Klimaschrank fährt dabei über einen Zeitraum von zwei Wochen ein sich veränderndes Luftfeuchtigkeitsprofil, wie in Abbildung 6.2 dargestellt, bei einer konstanten Temperatur von 23°C. Beginnend mit einer hohen relativen Luftfeuchtigkeit von 90% r.H., verringert sich die Luftfeuchtigkeit über die Zeit schrittweise bis zu einer relativen Luftfeuchtigkeit von 40% r.H.. Dann wird die relative Luftfeuchtigkeit sprunghaft wieder auf 90% r.H. angehoben. Anschließend für drei Tage sprunghaft wieder auf 40% r.H. gesenkt. Damit wird beabsichtigt, dass der sich über den Probenquerschnitt

einstellende Feuchtigkeitsgradient unregelmäßig ist. Zur Überwachung der tatsächlichen Klimawerte innerhalb des Klimaschrankes wird durch eine Durchführung der Klimasensor eines ortsfesten Klimaterminals in den Schrank eingebracht (siehe Abbildung 6.4)

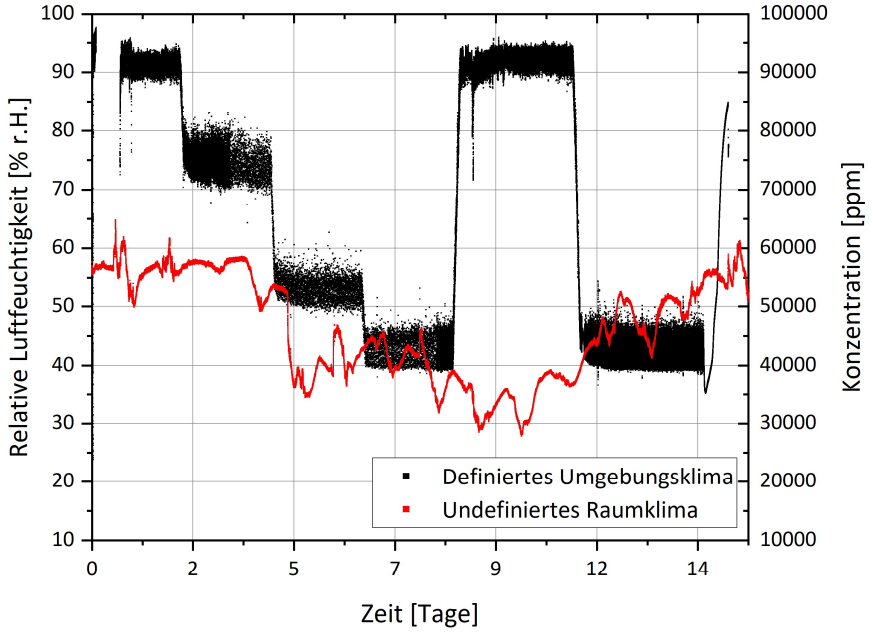


Abbildung 6.2: Relative Luftfeuchtigkeit der Laborumgebung (undefiniertes Raumklima) und des Klimaofens (definiertes Raumklima).

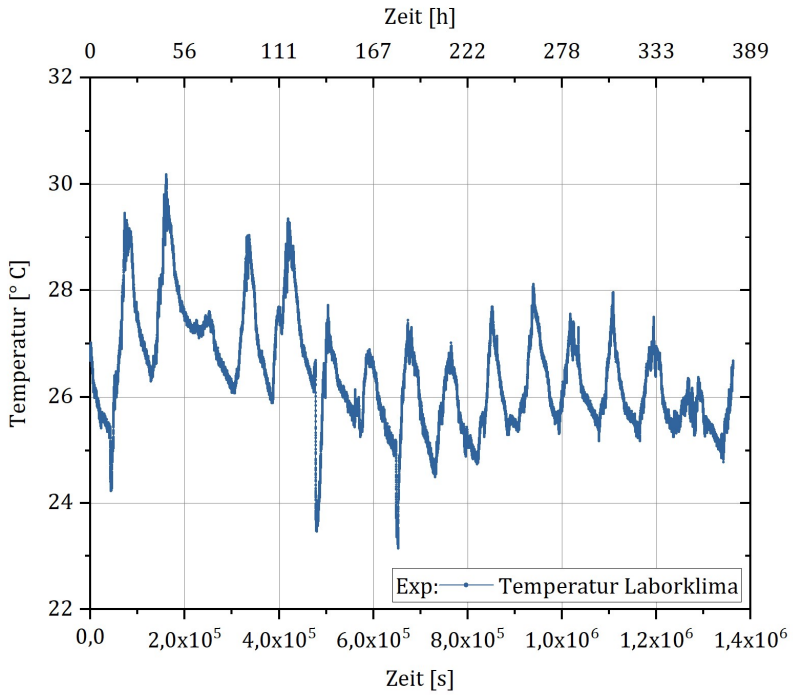


Abbildung 6.3: Darstellung des Temperaturverlaufs des im Labor vorherrschenden Klimas.

Bei der Durchführung der Konditionierung innerhalb des Klimaschrankes sind bereits nach vier Stunden Komplikationen aufgetreten (siehe Abbildung 6.2). Innerhalb des Klimaofens haben sich durch Kondensation Wassertröpfchen am Sensor gebildet. Diese haben vermutlich dazu geführt, dass der Sensor durch Kurzschluss den Kontakt zum I2c BUS verloren hat. Daraus resultierend war keine Datenaufnahme möglich. Es lässt sich schlussfolgern, dass die Sensoren *SHT31d* durch eine fehlende wasserdichte Kapselung nicht für den Einsatz in einer Messumgebung mit möglicher Tröpfchenbildung geeignet ist. Darüber hinaus ist auffällig, dass die Sensoren bei der Annäherung an die Grenze 90% relative Luftfeuchtigkeit bis zu 3% von dem am Klimaofen eingestellten Zielwert abwichen. Dieses Verhalten ist auch im zugehörigen Datenblatt des Sensors dokumentiert (siehe Anhang A5.2).

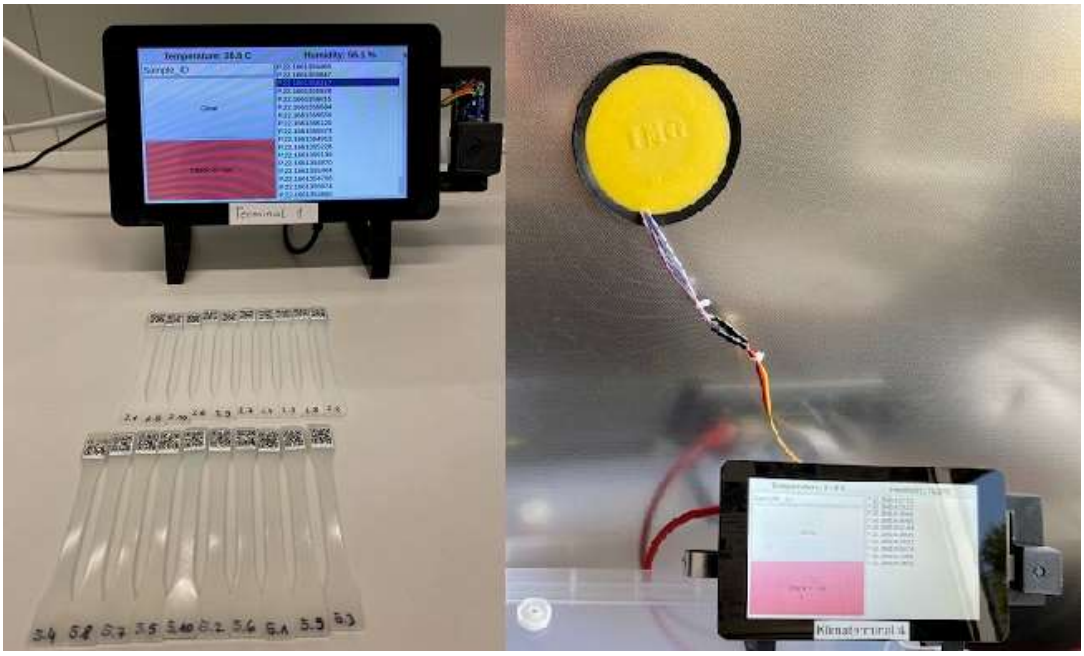


Abbildung 6.4: Ortsfeste Klimaterminals bei der Datenaufnahme: Datenaufnahme im Labor (links) und am Klimaofen mit in das Innere des Ofens verlängerter Sensorleitung (rechts).

6.3 Vergleich der Simulationsdaten mit den experimentell ermittelten Daten

Im Anschluss an die durchgeführten Versuche werden die aufgenommen Klima- und mechanischen Daten aus dem *Detact* System abgefragt. Alle Versuchsdaten sind als *Origin* Projekt in Anhang A6.2 abgelegt. Die aufgenommenen Klimadaten der Versuchsreihen liegen in einer hohen Dichte von jeweils einem relative Luftfeuchtigkeit- und Temperaturwert alle 6-7 Sekunden vor. Da anhand dieser Daten die Randbedingungen für die Simulation erstellt werden sollen, ist eine sinnvolle Reduktion der Datenmenge zur Verringerung der Rechendauer der Simulation durchzuführen. Zu diesem Zweck werden anhand der aufgenommenen Verläufe der Klimadaten Stützstellen definiert, zwischen denen Abaqus weitere Punkte interpolieren kann. Die Abbildungen 6.5 und 6.6 zeigen die Stützstellen im Profil der relativen Feuchtigkeit der Versuchsreihe eins, zwei und drei. Für die Reduktion der Datenmenge in äquidistanten Schritten ist ein *Python* Skript in Anhang A6.3 hinterlegt.

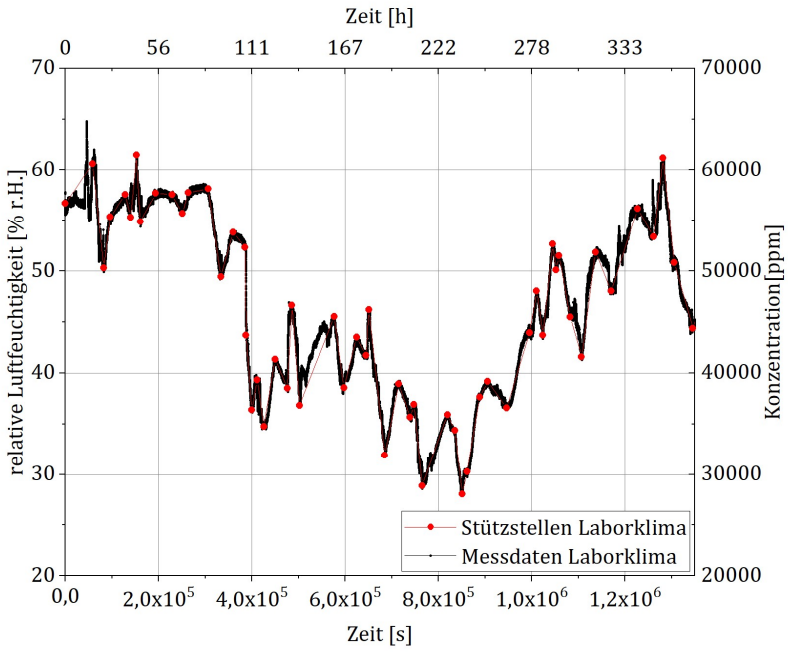


Abbildung 6.5: Stützstellen im Profil der relativen Luftfeuchtigkeit der Versuchsreihen zwei und drei.

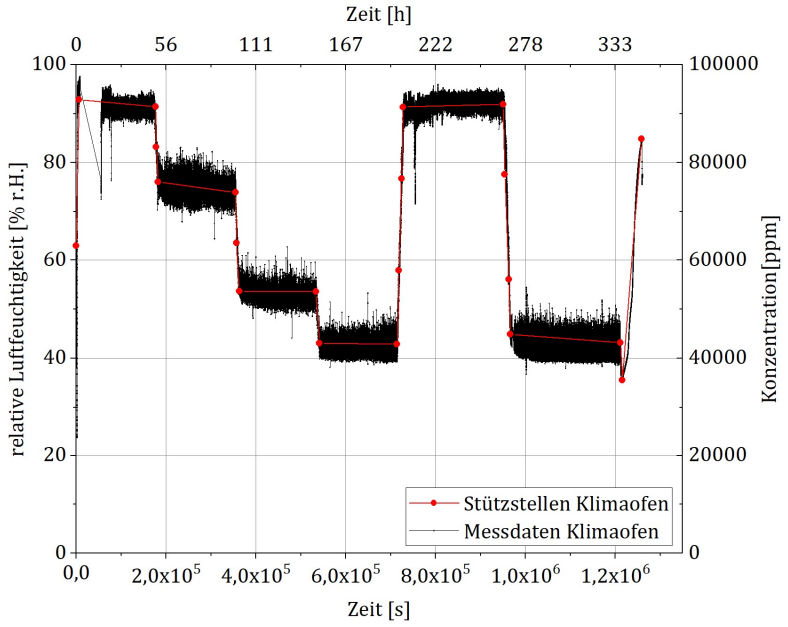


Abbildung 6.6: Stützstellen im Profil der relativen Feuchtigkeit der Versuchsreihe zwei.

Mit Hilfe der Stützstellen aus den Abbildungen 6.5 und 6.6 werden die Randbedingungen für die Simulation der Massendiffusion im 1BA Probekörper aus PA6 definiert. Die Ergebnisse der Simulation für die integrale Massenzunahme der Probekörper werden anschließend mit der real gemessenen Gewichtszunahme der Probekörper verglichen. Diese Vergleiche sind in den Abbildungen 6.7 bis 6.9 dargestellt. Die experimentell erfassten und im arithmetischen Mittel dargestellten Gewichte sind zu den jeweiligen Zeitpunkten als schwarzer Punkt dargestellt. Die schwarzen Linien über und unter den Messpunkten stellen die Standardabweichung der Gewichtsmessung dar. Die rote Linie kennzeichnet die simulierte Massenzunahme.

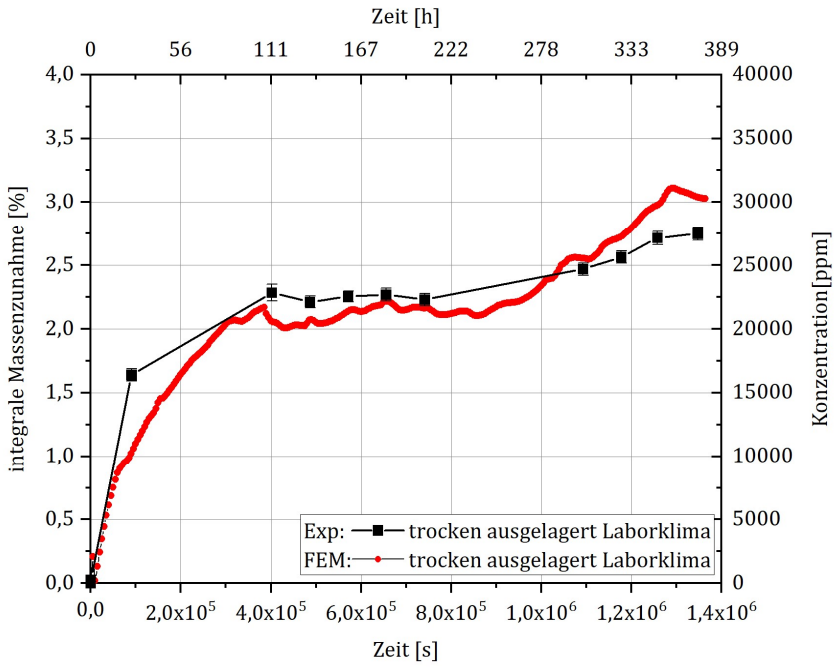


Abbildung 6.7: Vergleich der gemessenen Gewichtszunahme zur simulierten Gewichtszunahme der Proben aus Versuchsreihe eins.

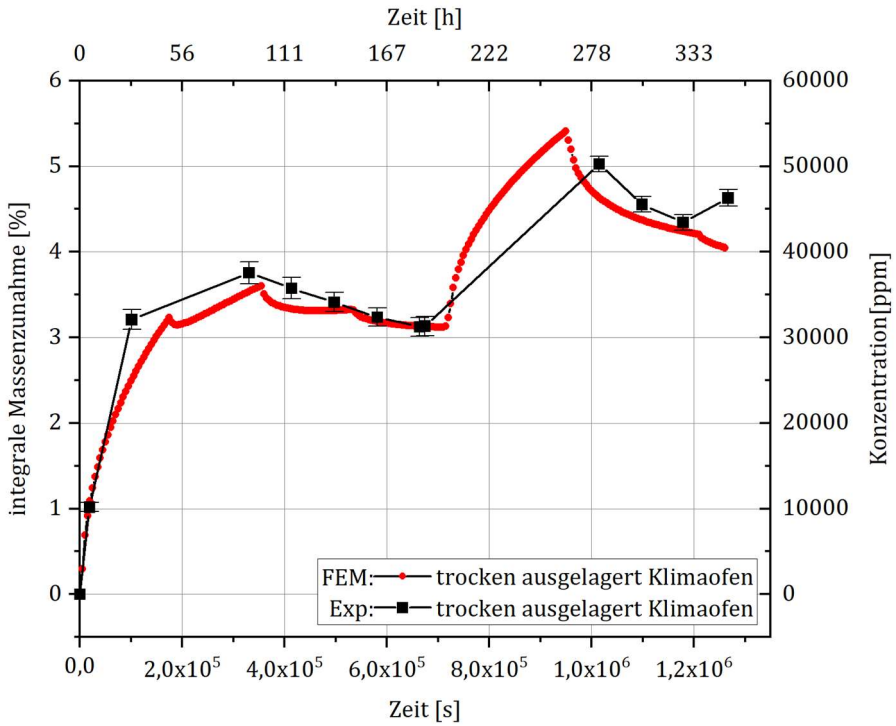


Abbildung 6.8: Vergleich der gemessenen Gewichtszunahme zur simulierten Gewichtszunahme der Proben aus Versuchsreihe zwei.

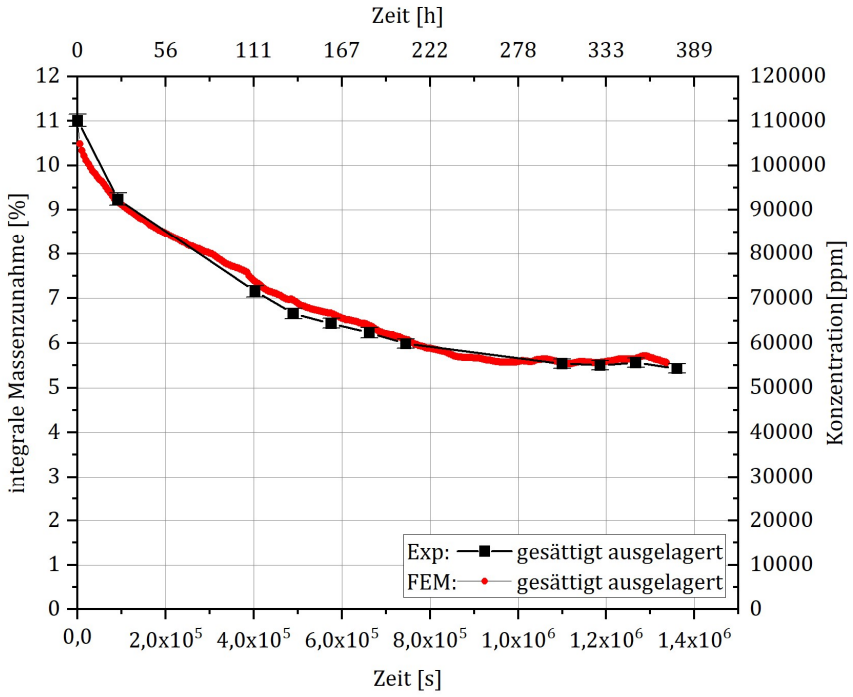


Abbildung 6.9: Vergleich der gemessenen Gewichtszunahme zur simulierten Gewichtszunahme der Proben aus Versuchsreihe drei.

Im zweiten Schritt der FE-Simulation wird, wie in Kapitel 2.2 beschrieben, der Massendiffusionssimulation ein einachsiger Zugversuch mittels statischer Spannungsanalyse nachgeschaltet. Bei diesem wird mit Hilfe von einem vordefinierten Feld die Konzentrationsverteilung des letzten Berechnungsschrittes der Massendiffusion als Feldvariable auf die nachgeschaltete Berechnung aufgeprägt, sodass die berechnete Feuchteverteilung über den Probekörperquerschnitt als Information zur FE-Berechnung der statischen Spannungsanalyse vorliegt. Durch die Verwendung von konzentrationsabhängiger E-Module und Querkontraktionszahlen in der Materialdefinition des statischen Spannungsanalyse-Modells berechnet *Abaqus*, unter Verwendung von Interpolation zwischen den Stützstellen der vorliegenden konzentrationsabhängigen mechanischen Größen, den resultierenden Spannungsverlauf innerhalb der Probe, unter Annahme der linearen Elastizität. Durch Umrechnung der so ermittelten Kraft-Verschiebungswerte zu Spannungs-Dehnungskurven, unter Verwendung der mittleren Probenquerschnitte, kann so auf den resultierenden E-Modul der inhomogenen Proben geschlossen werden. Diese berechneten E-Module können nun mit den an der Zwick/Roell aufgenommenen E-Modulen (siehe Anhang A6.4) verglichen werden. Abbildung 6.10 zeigt den Vergleich der gemittelten, gemessenen E-Module zu den simulierten E-Modulen der Versuchsreihen eins, zwei und drei.

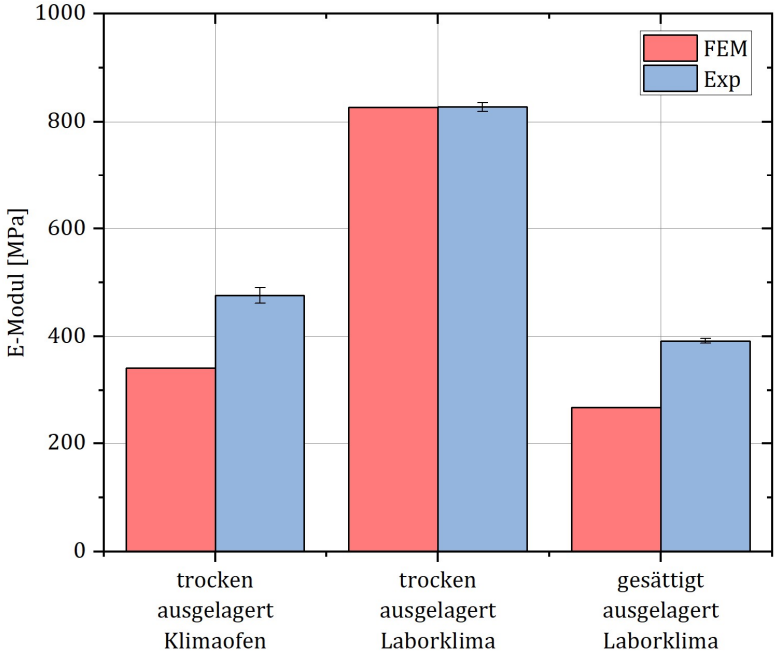


Abbildung 6.10: Gegenüberstellung der gemessenen und simulierten E-Module.

6.4 Diskussion der Ergebnisse

Beim Betrachten der gemessenen Massenzunahme der Proben im arithmetischen Mittel, im Vergleich zur simulierten Massenzunahme ist deutlich zu erkennen, dass die Simulation den Verlauf der realen Massenzunahme gut abbilden kann. Der Verlauf der Kurven weicht maximal um 0,63 Gew.% von den gemessenen Werten ab. Die Abweichung in Abbildung 6.7, welche die Kurven für Versuchsreihe zwei zwischen Stunde 195 und 280 darstellt, sind auf Grund fehlender Messpunkte nicht vollständig als aussagekräftig einzustufen. Die aufgenommenen E-Module stimmen lediglich in einem Fall (Versuchsreihe zwei) gut überein. Der E-Modul der anderen beiden Versuchsreihen wird durch die Simulation um etwa 30% unterschätzt. Die innerhalb der Simulation angesetzten Diffusionskoeffizienten sind für 23°C validiert. Diese konstante Temperaturrandbedingung liegt jedoch lediglich in der Versuchsreihe zwei, innerhalb des Klimaofens vor. Die im Laborklima ausgelagerten Proben, wie in Abbildung 6.3 zu sehen, waren zumeist leicht höheren Umgebungstemperaturen von bis zu 30°C ausgesetzt. Diese geringen Temperaturschwankungen und die damit einhergehende, wenn auch als gering einzuschätzende, Veränderung der Diffusionskoeffizienten ist im Simulationsmodell in dieser Form momentan noch nicht berücksichtigt. Die gute Übereinstimmung der simulativen Ergebnisse mit den Messwerten im Bereich der intergralen Massenzunahme legen den Schluss nahe, dass die Temperatur bei den Sorptionsprozessen eine untergeordnete Rolle spielt. Diese Vermutung ist in Zukunft durch eine Erweiterung des Simulationsmodells sowie weiteren Versuchsreihen zu überprüfen.

7 Zusammenfassung und Fazit

Im Rahmen dieser Arbeit wurde ein vernetztes System (Monitoringumgebung) aus datenerhebenden Stationen entwickelt und an das *Symate Detact* Industrie 4.0 System über das *MQTT*-Protokoll angegliedert. Die in Versuchsreihen aufgenommenen und im *Detact* System vorgehaltenen Daten wurden abschließend als zeitlich veränderliche Randbedingung in einem FEM-Simulationsmodell genutzt und die numerisch berechneten Ergebnisse mit experimentell ermittelten Ergebnissen verglichen. Die Monitoringumgebung besteht aus vier ortsfesten und zwei mobilen Terminals zur Überwachung von Klimadaten, einer Software für das Einpflegen erfasster Daten aus einer Zwick/Roell Universalprüfmaschine sowie einer Software zur Erzeugung eindeutiger, innerhalb dieser Arbeit spezifizierter, Probenbezeichner (Sample IDs). Die Komponenten werden mit einer in dieser Arbeit entwickelten Software betrieben, die die Einstellung der einzelnen Systeme, die Datenaufnahme, Datenaufbereitung und Datenweitergabe an das *Detact* System sowie eine einfache Benutzerführung leistet. Die entwickelte Software ist innerhalb der Arbeit umfänglich beschrieben. Die Leistungsfähigkeit der Monitoringumgebung ist in drei Versuchsreihen überprüft worden. Auch eine Anleitung zur wesentlichen Bedienung und Einstellung der einzelnen Komponenten der Monitoringumgebung wurde erstellt und in den Anhängen A5.7, A5.16, A5.20 und A5.25 der Arbeit angefügt. Die Monitoringumgebung ist grundsätzlich in der Lage die an sie in der Aufgabenstellung spezifizierten Anforderungen zu erfüllen. Durch die im Rahmen dieser Arbeit geschaffenen Definitionen für die Kommunikation über *MQTT* mit dem *Detact* System lassen sich weitere (auch ältere) Maschinen an das *Detact* System angliedern. Mit Hilfe der Monitoringumgebung wurden drei Versuchsreihen für die Validierung des in Kapitel 3.2 beschriebenen FEM-Simulationsmodells von Sambale [8] in Bezug auf zeitlich veränderliche Umgebungskonzentrationen von Wasser auf PA6 Probekörper durchgeführt.

Die Versuchsreihen zeigen deutlich, dass das FEM-Simulationsmodell in der Lage ist den zeitlichen Verlauf sowie den Betrag der integralen Massenzunahme von Wasser in PA6 auf Grund von Sorptions- und Desorptionsprozessen mit einer maximalen Abweichung von 0,63 Massenprozent abzubilden. Eine hinreichend genaue Abschätzung des E-Modules durch das FEM-Modell ist zum Zeitpunkt des

Abschlusses der Arbeit nicht gegeben. Eine Erweiterung des FEM-Modells zur Berücksichtigung sich zeitlich verändernder Temperaturrandbedingungen (welche die Monitoringumgebung bereits liefern kann) stellt eine Möglichkeit zu weiterer Verbesserung der numerisch berechneten Ergebnisse da. Die einzelnen Komponenten der Monitoringumgebung haben sich als funktional herausgestellt. Dies gilt insbesondere für die ortsfesten Terminals. Es handelt sich bei den Komponenten jeweils um neu entwickelte Produkte, die durchaus an einigen Stellen Verbesserungsbedarf aufweisen und erweitert werden sollten. Mögliche Gesichtspunkte zur Weiterentwicklung der Monitoringumgebung werden in Kapitel 8 aufgeführt.

8 Verbesserungsvorschläge

- Austausch der an den ortfesten Terminals verwendeten Kameras durch professionelle USB-Handscanner, da sich die Zuverlässigkeit der Kameras bei unterschiedlichen Etikett Größen und Beleuchtungsverhältnissen als ungenügend herausgestellt hat.
- Klimaterminals mit feuchtigkeitsbeständigen T/H-Sensoren ausstatten.
- Einbau der ortsfesten Terminals in feste Gehäuse zur Vorbeugung von Beschädigungen und Verschmutzungen.
- Erstellung weiterer ortsfester Terminals, da sich diese als gut zu nutzendes und zuverlässiges Werkzeug herausgestellt haben.
- Erweiterung der Gehäuse der mobilen Terminals um Schnapphakenverbindungen für einen besseren Formschluss zwischen dem Gehäusegrundköper und dem Gehäusedeckel.
- Erweiterung der Monitoringumgebung um eine Waage zur automatischen Übermittlung von Wiegungen an das *Detact* System.
- Die ID Vergabe Software auf einem explizit für die Erstellung von Probeetiketten vorgesehenen Rechner mit Laserbeschriftungsgerät innerhalb des Labors betreiben.
- Entwicklung einer *Android*-App für die Sample ID Vergabe.
- Beim Start der einzelnen Komponenten der Monitoringumgebung überprüfen, ob die Systemzeit tatsächlich mit *NTP* synchronisiert wurde.
- Anbindung des *Abaqus* Simulationssystems an das *Detact* System für die direkte Rückführung der simulativ erhaltenen Daten.
- Automatische Generierung von Stützstellen aus den aufgenommenen Klimadaten für das einpflegen der Randbedingungen in das Abaqus Simulationssystem.
- Berücksichtigung der sich zeitlich ändernden Umgebungstemperatur innerhalb des FEM-Simulationsmodells.

9 Literatur

[1] *Bauer; Brinkmann; Osswald et al.:* Saechtling Kunststoff Taschenbuch. Hanser, München, 2013.

[2] *Kohan, M.I. (Hrsg.):* Nylon plastics handbook – With 169 tables. Hanser, Munich, 1995.

[3] *Sambale, A.; Kurkowski, M.; Stommel, M.:* Determination of moisture gradients in polyamide 6 using StepScan DSC. *In:* Thermochimica Acta (2019), S. 150-156. https://doi.org/10.1016/j.tca.2018.12.011.

[4] *LanXess:* Durethan B 30 S and B 31 SK. LanXess, https://techcenter.lanxess.com/scp/americas/en/docguard/PIB_Durethan_B30S_and_B31SK.pdf?docId=76986 [Zugriff am: 27.10.2022].

[5] *Binsack, R.:* Technische Thermoplaste - Polyamide, Kunststoff-Handbuch. Neuausgabe - Becker/Braun3/4, Hanser, München, 1998.

[6] *Emde, J.:* Experimentelle Charakterisierung und FE-Modellierung des Sorptions- und Quellverhaltens von Polyamid 6 in Wasser. Dortmund, Tu Dortmund, Master Thesis, 2020.

[7] *Sambale, A.K.; Maisl, M.; Herrmann, H.G. et al.:* Characterisation and Modelling of Moisture Gradients in Polyamide 6. *In:* Polymers.

[8] *Sambale, A.:* Beitrag zur Charakterisierung und Berechnung von Feuchtigkeitsverteilungen in Polyamid 6. Dresden, Technische Univerität Dresden, Dissetation (eingereicht), 2022.

[9] *Sambale, A.K.; Stanko, M.; Emde, J. et al.:* Characterisation and FE Modelling of the Sorption and Swelling Behaviour of Polyamide 6 in Water. *In:* Polymers.

[10] *Zschech, C.; Sambale A.; Henn, E. et al.:* Monitoring von Werkstoffkenngrößen entlang des gesamten Produktlebenszyklus, Vortrag, 2022.

[11] *Symate GmbH:* Detact - Künstliche Intelligenz für die digitale Fertigung. Symate GmbH, 2022, https://www.detact.com/ [Zugriff am: 29.09.2022].

[12] *Sallat, M.:* Das Anwendungsprotokoll MQTT im Internet of Things. Offenburg, Hochschuhle Offenburg, Bachelor Thesis, 2018.

[13] *Atzori, L.; Iera, A.; Morabito, G.:* Understanding the Internet of Things: definition, potentials, and societal role of a fast evolving paradigm. *In:* Ad Hoc Networks 56 (2017), S. 122-140. https://doi.org/10.1016/j.adhoc.2016.12.004.

[14] Internet der Dinge, 2022, https://www.tenmedia.de/de/glossar/internet-der-dinge [Zugriff am: 15.10.2022].

[15] Was bedeutet IoT und was bedeutet IIoT?, 2021, https://www.copadata.com/de/produkt/platform-editorial-content/was-sind-iot-iiot/ [Zugriff am: 17.10.2022].

[16] *Botthof A.; Hartmann, E.A.:* Zukunft der Arbeit in Industrie 4.0. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.
[17] Was ist Industrie 4.0?, 2022, https://www.plattform-i40.de/IP/Navigation/DE/Industrie40/WasIndustrie40/was-ist-industrie-40.html [Zugriff am: 15.10.2022].
[18] *DIN e.V. DKE:* Deutsche Normungsroadmap Industrie 4.0 – Version 4. DIN e.V. DKE, 2022, https://www.din.de/de/forschung-und-innovation/themen/industrie4-0/roadmap-industrie40-62178 [Zugriff am: 27.10.2022].
[19] *gambit:* Industrie 4.0. gambit, 2022, https://www.gambit.de/wiki/industrie-4-0/ [Zugriff am: 15.10.2022].
[20] *Heger, J.; El Abdine, M.Z.; Sekar, S. et al.:* Simulation in Produktion und Logistik 2021 – Erlangen, 15.-17. September 2021, ASIM-MitteilungAM 177, Cuvillier Verlag, Göttingen, 2021.
[21] *Stadler, M.L.:* Digital Twin, 2021, https://mindsquare.de/knowhow/digital-twin/#vorteile-von-digital-twins [Zugriff am: 15.10.2022].
[22] *Fertien, S.; Lang, J.; Immerman, D.:* Digital Twin: A Primer for Industrial Enterprises, 2019, https://www.ptc.com/-/media/Files/PDFs/IoT/digital_twin_industrial-enterprises-6-11-19.pdf [Zugriff am: 15.10.2022].
[23] *Vertommen, I.:* Digital twins for the water sector - KWR, 2022, https://www.kwrwater.nl/en/projecten/digital-twins-for-the-water-sector/ [Zugriff am: 15.10.2022].
[24] *Peters, D.:* Wie wichtig Schnittstellen für die Digitalisierung von Geschäftsprozessen sind, 2021, https://www.hamburger-software.de/blog/schnittstellen-fuer-digitalisierung-wichtig-fuer-geschaeftsprozesse/ [Zugriff am: 18.10.2022].
[25] *Kurose, J.F.; Ross, K.W.:* Computernetzwerke – Der Top-Down-Ansatz, IT - Informatik, Pearson Studium, München, 2008.
[26] *Meinel, C.:* Internetworking – Technische Grundlagen und Anwendungen, SpringerLink Bücher, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
[27] *Baun, C.:* Computernetze Kompakt, IT Kompakt Ser, Springer Berlin / Heidelberg, Berlin, Heidelberg, 2015.
[28] *HiveMQ:* Introducing the MQTT Protocol - MQTT Essentials: Part 1. HiveMQ, 2022, https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/ [Zugriff am: 16.10.2022].
[29] *IONOS:* Was ist HTTP? IONOS, 2022, https://www.ionos.de/digitalguide/hosting/hosting-technik/was-ist-http/ [Zugriff am: 16.10.2022].
[30] *Guzel, B.:* HTTP-Header für Dummies, 2021, https://code.tutsplus.com/de/tutorials/http-headers-for-dummies--net-8039 [Zugriff am: 16.10.2022].
[31] *ComputerWeekly.de:* Was ist REST API (RESTful API), 2022, https://www.computerweekly.com/de/definition/RESTful-API [Zugriff am: 16.10.2022].
[32] *Al-Fuqaha, A.; Guizani, M.; Mohammadi, M. et al.:* Internet of Things: A Survey

on Enabling Technologies, Protocols, and Applications. *In:* IEEE Communications Surveys & Tutorials 17 (2015), Heft 4, S. 2347-2376. https://doi.org/10.1109/COMST.2015.2444095.
[33] *Trojan, W.:* Das MQTT-Praxisbuch. Elektor Verlag GmbH, An Elector Publication, Elektor-Verlag GmbH, Aachen, 2017.
[34] Quality of Service (QoS) 0,1, & 2 MQTT Essentials: Part 6, 2022, https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/ [Zugriff am: 16.10.2022].
[35] *OPC Foundation:* What is OPC?, 2017, https://opcfoundation.org/about/what-is-opc/ [Zugriff am: 16.10.2022].
[36] *Helmbrecht-Schaar, A.:* IIoT Protocols: OPC UA vs. MQTT Sparkplug, 2022, https://www.hivemq.com/iiot-protocols-opc-ua-mqtt-sparkplug-comparison/ [Zugriff am: 17.10.2022].
[37] *Behara, G.K.:* Open Source IoT Middleware for the Integration of Enterprise Applications, 2019, https://www.opensourceforu.com/2019/10/a-primer-on-open-source-iot-middleware-for-the-integration-of-enterprise-applications/ [Zugriff am: 17.10.2022].
[38] Eclipse Ditto documentation overview – Eclipse Ditto a digital twin framework, 2022, https://www.eclipse.org/ditto/intro-overview.html [Zugriff am: 17.10.2022].
[39] *Siemens Digital Industries Software:* Mindsphere. Siemens Digital Industries Software, 2022, https://www.plm.automation.siemens.com/global/en/products/mindsphere/ [Zugriff am: 17.10.2022].
[40] *PTC:* Thoughts on the Digital Transformation of the Remote Monitoring Medical Device Industry | PTC, 2022, https://www.ptc.com/en/resources/iiot/brochure/thingworx-overview [Zugriff am: 17.10.2022].
[41] *Klein, U.:* Wetterstation Test Check: Die besten Wetterstationen 2022, 2021, https://www.homeandsmart.de/wetterstation-test-vergleich [Zugriff am: 18.10.2022].
[42] *Wörner, N.:* Digitale Messung im Schaltschrank: Kondensat vermeiden, 2020, https://www.elektroniknet.de/messen-testen/sensorik/kondensat-vermeiden.175290.html [Zugriff am: 18.10.2022].
[43] *Eberle, S.:* MessprinzipienTemperatur/Feuchte. G. Lufft Mess- und Regeltechnik GmbH, https://www.lufft.com/fileadmin/lufft.com/05_service/de_Messprinzipien_Temperatur_Feuchte.pdf [Zugriff am: 23.10.2022].
[44] *Abacha, N.; Kubouchi, M.; Sakai, T.:* Diffusion behavior of water in polyamide 6 organoclay nanocomposites. *In:* Express Polymer Letters 3 (2009), Heft 4, S. 245-255. https://doi.org/10.3144/expresspolymlett.2009.31.
[45] *Vlasveld, D.; Groenewold, J.; Bersee, H. et al.:* Moisture absorption in polyamide-6 silicate nanocomposites and its influence on the mechanical properties. *In:* Polymer 46 (2005), Heft 26, S. 12567-12576. https://doi.org/10.1016/j.polymer.2005.10.096.
[46] *DIN EN ISO:* Kunststoffe - Vielzweckprobekörper, DIN EN ISO, Norm.

[47] DIN EN ISO 527-2 Typ 1BA - Q-tec GmbH, 2022, https://q-tec-gmbh.de/en/products/cutting-die/din-en-iso-527/din-en-iso-527-2-typ-1ba/ [Zugriff am: 19.10.2022].
[48] *Sambale, A.K.:* Anleitung zum Aufbau des Berechnungsmodells in Abaqus/CAE (Version 2021), Anleitung.
[49] 9834-8:2005: Information technology - Open Systems Interconnection - Remote Procedure Call (RPC).
[50] *biancahoegel.de:* Network Time Protocol, 2022, https://www.biancahoegel.de/computer/netz/ntp.html [Zugriff am: 27.10.2022].
[51] *Lehmann, L.; Schönfeld, D.:* QR-Codes, https://tu-dresden.de/ing/informatik/sya/ps/ressourcen/dateien/studium/materialien/mat_kanalkodierung/QR_Code_pdf?lang=de [Zugriff am: 29.09.2022].
[52] *PyPI:* qrcode 7.3.1. PyPI, 2022, https://pypi.org/project/qrcode/ [Zugriff am: 29.09.2022].
[53] *Acadamic.com:* Unix-Timestamp, 2022, https://de-academic.com/dic.nsf/dewiki/1438060 [Zugriff am: 27.10.2022].
[54] File:Ethernet.svg - Wikimedia Commons, 2022, https://commons.wikimedia.org/wiki/File:Ethernet.svg [Zugriff am: 14.10.2022].
[55] Wifi Signal Vector SVG Icon (10) - SVG Repo, 2022, https://www.svgrepo.com/svg/95706/wifi-signal [Zugriff am: 14.10.2022].
[56] Introducing JSON – EMCA-040 The JSON Data interchange Standard, 2022, https://www.json.org/json-en.html [Zugriff am: 29.09.2022].
[57] *Luber, S.:* Was ist JSON?, 2018, https://www.cloudcomputing-insider.de/was-ist-json-a-704909/ [Zugriff am: 29.09.2022].
[58] *Oracle:* JSON Developer's Guide, 2022, https://docs.oracle.com/en/database/oracle/oracle-database/21/adjsn/json-data.html#GUID-FBC22D72-AA64-4B0A-92A2-837B32902E2C [Zugriff am: 29.09.2022].
[59] *Upton, E.* Happy Birthday to us [online]. *In:* Raspberry Pi, 2022 [Zugriff am: 29.09.2022], https://www.raspberrypi.com/news/happy-birthday-to-us-3/.
[60] *Pini, A.:* Warum der Inter-Integrated Circuit (I2C)-Bus den Anschluss von ICs so einfach macht - und wie man ihn verwendet, 2020, https://www.digikey.de/de/articles/why-the-inter-integrated-circuit-bus-makes-connecting-ics-so-easy [Zugriff am: 29.09.2022].
[61] *Rixecker, K.:* Die beliebtesten Programmiersprachen: Python überholt C#, 2018, https://t3n.de/news/beliebtesten-programmiersprachen-979869/ [Zugriff am: 29.09.2022].
[62] QThread - Qt for Python, 2022, https://doc.qt.io/qtforpython/PySide6/QtCore/QThread.html [Zugriff am: 29.09.2022].
[63] Using Qt Designer — PyQt Documentation v5.15.4, 2022, https://www.riverbankcomputing.com/static/Docs/PyQt5/designer.html [Zugriff am: 29.09.2022].
[64] Signals & Slots - Qt for Python, 2022,

https://doc.qt.io/qtforpython/overviews/signalsandslots.html [Zugriff am: 29.09.2022].
[65] *Python Software Foundation:* configparser - Configuration file Parser. Python Software Foundation, 2022, https://docs.python.org/3/library/configparser.html [Zugriff am: 27.10.2022].
[66] *PyPI:* paho-mqtt 1.6.1. PyPI, 2022, https://pypi.org/project/paho-mqtt/ [Zugriff am: 04.10.2022].
[67] QThread Class | Qt Core 5.15.10, 2022, https://doc.qt.io/qt-5/qthread.html [Zugriff am: 02.10.2022].
[68] QApplication — Qt for Python, 2020, https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QApplication.html#more [Zugriff am: 30.09.2022].
[69] QCoreApplication — Qt for Python, 2020, https://doc.qt.io/qtforpython-5/PySide2/QtCore/QCoreApplication.html [Zugriff am: 30.09.2022].
[70] QEventLoop — Qt for Python, 2020, https://doc.qt.io/qtforpython-5/PySide2/QtCore/QEventLoop.html#detailed-description [Zugriff am: 30.09.2022].
[71] *Bodnar, J.:* Events and signals in PyQt5, 2022, https://zetcode.com/gui/pyqt5/eventssignals/ [Zugriff am: 30.09.2022].
[72] datetime — Basic date and time types — Python 3.10.7 documentation, 2022, https://docs.python.org/3/library/datetime.html [Zugriff am: 30.09.2022].
[73] json — JSON encoder and decoder — Python 3.10.7 documentation, 2022, https://docs.python.org/3/library/json.html [Zugriff am: 30.09.2022].
[74] *lady ada:* Adafruit SHT31-D Temperature & Humidity Sensor Breakout, 2021.
[75] Raspberry Pi Camera Module, https://docs.rs-online.com/3b9b/0900766b814db308.pdf [Zugriff am: 30.10.2022].
[76] *sertronics:* offizielles Raspberry Pi 7 Display mit kapazitiven Touchscreen. sertronics, 2022, https://www.berrybase.de/Pixelpdfdata/Articlepdf/id/3773/onumber/RPIO-7LCD [Zugriff am: 27.10.2022].
[77] *josephtylerjones:* Minimalist Stand for the Raspberry Pi 7" Touch Display by josephtylerjones. Thingiverse, 2022, https://www.thingiverse.com/thing:4772231 [Zugriff am: 01.10.2022].
[78] *VGer:* Raspberry pi camera case/enclosure by VGer. Thingiverse.com, 2022, https://www.thingiverse.com/thing:92208 [Zugriff am: 01.10.2022].
[79] Raspberry Pi Documentation - Raspberry Pi OS, 2022, https://www.raspberrypi.com/documentation/computers/os.html [Zugriff am: 01.10.2022].
[80] random — Generate pseudo-random numbers — Python 3.10.7 documentation, 2022, https://docs.python.org/3/library/random.html [Zugriff am: 04.10.2022].
[81] Logging HOWTO — Python 3.10.7 documentation, 2022, https://docs.python.org/3/howto/logging.html [Zugriff am: 04.10.2022].
[82] sys — System-specific parameters and functions — Python 3.10.7 documentation, 2022, https://docs.python.org/3/library/sys.html [Zugriff am: 04.10.2022].

[83] csv — CSV File Reading and Writing — Python 3.10.7 documentation, 2022, https://docs.python.org/3/library/csv.html [Zugriff am: 04.10.2022].
[84] QtCore — PyQt 5.9 Reference Guide, 2017, https://docs.huihoo.com/pyqt/PyQt5/QtCore.html [Zugriff am: 04.10.2022].
[85] QtGui — PyQt 5.9 Reference Guide, 2017, https://docs.huihoo.com/pyqt/PyQt5/QtGui.html [Zugriff am: 04.10.2022].
[86] *Fitzpatrick, M.:* PyQt5 Widgets, 2019, https://www.pythonguis.com/tutorials/pyqt-basic-widgets/ [Zugriff am: 04.10.2022].
[87] Signal - Qt for Python, 2022, https://doc.qt.io/qtforpython/PySide6/QtCore/Signal.html [Zugriff am: 02.10.2022].
[88] Slot — Qt for Python, 2020, https://doc.qt.io/qtforpython-5/PySide2/QtCore/Slot.html [Zugriff am: 02.10.2022].
[89] time — Time access and conversions — Python 3.10.7 documentation, 2022, https://docs.python.org/3/library/time.html [Zugriff am: 02.10.2022].
[90] *PyPI:* board 1.0. PyPI, 2022, https://pypi.org/project/board/ [Zugriff am: 02.10.2022].
[91] Adafruit SHT31D Library 1.0 documentation, 2022, https://docs.circuitpython.org/projects/sht31d/en/latest/api.html [Zugriff am: 02.10.2022].
[92] *PyPI:* opencv-python 4.6.0.66. PyPI, 2022, https://pypi.org/project/opencv-python/ [Zugriff am: 03.10.2022].
[93] *PyPI:* pyzbar 0.1.9. PyPI, 2022, https://pypi.org/project/pyzbar/ [Zugriff am: 03.10.2022].
[94] *PyPI:* imutils 0.5.4. PyPI, 2022, https://pypi.org/project/imutils/ [Zugriff am: 03.10.2022].
[95] Funkobst [online]. *In:* Heise, 2017 [Zugriff am: 09.10.2022], https://www.heise.de/select/ct/2017/7/1490889430179451.
[96] *Raspberry Pi Foundation:* Teach, learn, and make with the Raspberry Pi Foundation. Raspberry Pi Foundation, 2022, https://www.raspberrypi.org/ [Zugriff am: 09.10.2022].
[97] Android Icon Logo Mobile App - android logo png - Unlimited Download. cleanpng.com, 2022 [Zugriff am: 09.10.2022].
[98] *PyPI:* RPi.GPIO 0.7.1. PyPI, 2022, https://pypi.org/project/RPi.GPIO/ [Zugriff am: 11.10.2022].
[99] UPS HAT (C) - Waveshare Wiki, 2022, https://www.waveshare.com/wiki/UPS_HAT_(C) [Zugriff am: 11.10.2022].
[100] AsyncTask Android Developers, 2022, https://developer.android.com/reference/android/os/AsyncTask [Zugriff am: 13.10.2022].
[101] Introduction to activities - Android Developers, 2022, https://developer.android.com/guide/components/activities/intro-activities [Zugriff am: 13.10.2022].
[102] Activity Android Developers, 2022, https://developer.android.com/reference/android/app/Activity [Zugriff am:

13.10.2022].
[103] Appcompat Android Developers, 2022, https://developer.android.com/jetpack/androidx/releases/appcompat [Zugriff am: 13.10.2022].
[104] Maven Repository: com.google.android.material material, 2022, https://mvnrepository.com/artifact/com.google.android.material/material?repo=google [Zugriff am: 13.10.2022].
[105] Constraintlayout Android Developers, 2022, https://developer.android.com/jetpack/androidx/releases/constraintlayout [Zugriff am: 13.10.2022].
[106] Maven Repository: androidx.legacy legacy-support-v4 1.0.0-alpha1, 2022, https://mvnrepository.com/artifact/androidx.legacy/legacy-support-v4/1.0.0-alpha1 [Zugriff am: 13.10.2022].
[107] *Craggs, I.:* Eclipse Paho | The Eclipse Foundation, 2022, https://www.eclipse.org/paho/index.php?page=clients/android/index.php [Zugriff am: 13.10.2022].
[108] Localbroadcastmanager Android Developers, 2022, https://developer.android.com/jetpack/androidx/releases/localbroadcastmanager [Zugriff am: 13.10.2022].
[109] Maven Repository: com.budiyev.android code-scanner 2.1.0, 2022, https://mvnrepository.com/artifact/com.budiyev.android/code-scanner/2.1.0 [Zugriff am: 13.10.2022].
[110] *Zwick/Roell:* testXpert III – Neue Prüfsoftware mit intuitiver und workfloworientierter Bedienung. Zwick/Roell, 2022, https://www.zwickroell.com/de/news-events/news/testxpert-iii/ [Zugriff am: 04.10.2022].
[111] *Zwick/Roell:* zwickiLine. Zwick/Roell, 2022, https://www.zwickroell.com/de/produkte/statische-material-pruefmaschinen/universalpruefmaschinen-fuer-statische-anwendungen/zwickiline/#c3366 [Zugriff am: 04.10.2022].
[112] *Zwick/Roell:* Produktinformation Material-Prüfmaschinen zwickiLine Z0.5 bis Z5.0. Zwick/Roell, https://www.zwickroell.com/fileadmin/content/Files/SharePoint/user_upload/PI_DE/02_396_Material_Pruefmaschine_zwickiLine_Z0_5_bis_Z5_0_PI_DE.pdf [Zugriff am: 04.10.2022].
[113] KP1401 Tischscanner, https://cdn-reichelt.de/documents/datenblatt/EB00/KAPTUR_KP1401_DB-DE.pdf [Zugriff am: 07.10.2022].
[114] os — Miscellaneous operating system interfaces — Python 3.10.7 documentation, 2022, https://docs.python.org/3/library/os.html [Zugriff am: 06.10.2022].
[115] pandas - Python Data Analysis Library, 2022, https://pandas.pydata.org/ [Zugriff am: 06.10.2022].
[116] QFileSystemWatcher — Qt for Python, 2020, https://doc.qt.io/qtforpython-5/PySide2/QtCore/QFileSystemWatcher.html [Zugriff am: 06.10.2022].
[117] *Lundh, F.; Clark, P.:* Python Pillow, 2022, https://python-pillow.org/ [Zugriff

am: 07.10.2022].
[118] Qt Print Support 6.4.0, 2022, https://doc.qt.io/qt-6/qtprintsupport-index.html [Zugriff am: 07.10.2022].

10 Anhang

Folgend sind die dem Dokument angehängten Anlagen aufgeführt. Es wird die Anlagennummer, Anlagenbezeichnung und der Anlagendateiname angegeben.

Bezeichnung:	**Beschreibung:**	**Dateiname:**
A5.1	Datenblatt Raspberry Pi 4 Model B	*A5_1 raspberry-pi-4-datasheet.pdf*
A5.2	Datenblatt Sensirion SHT31	*A5_2 Sensirion_Humidity_SHT3x_Datasheet.pdf*
A5.3	Datenblatt Raspberry Pi Kameramodule v2.1	*A5_3 Raspberry Pi Kameramodule v2.1 Datenblatt.pdf*
A5.4	Datenblatt Raspberry Pi 7“ Touchdisplay	*A5_4 offizielles-Raspberry-Pi-7-Display-mit-kapazitiven-Touchscreen_Datenblatt.pdf*
A5.5	Technische Zeichnung Sensorträger	*A5_5 Zeichnung ortsfestes Terminal Sensorträger.pdf*
A5.6	Datenblatt Raspberry Pi USB-C Netzteil	*A5_6 Raspberry Pi Netzteil Datenblatt.pdf*
A5.7	Bedienungsanleitung Ortsfeste Terminals	*A5_7 Bedienungsanleitung*

		ortsfeste Terminals.pdf
A5.8	PyCharm Softwareprojekt der ortsfesten und mobilen Terminals	*A5_8 Probenmonitoring.zip*
A5.9	Datenblatt Raspberry Pi Zero W	*A5_9 Datenblatt_PiZero.pdf*
A5.10	Datenblatt Waveshare UPS HAT (C)	*A5_10 Waveshare UPS HAT C Datenblatt.pdf*
A5.11	Datenblatt Joy-It 2x16 Zeilen LCD-Display	*A5_11 Joy-IT 2x16 LCD Datenblatt.pdf*
A5.12	Datenblatt Taster MSC18	*A5_12 MSC18 TASTER Datenblatt.pdf*
A5.13	Datenblatt Netzteil Voltcraft DC 5V 2,5A	*A5_13 Micro USB Netzteil Daatenblatt.pdf*
A5.14	Technische Zeichnung Grundgehäuse mobile Terminals	*A5_14 Zeichnung mobiles Terminal Grundgehäuse.pdf*
A5.15	Technische Zeichnung Gehäusedeckel mobile Terminals	*A5_15 Zeichnung mobiles Terminal Gehäuse Deckel.pdf*
A5.16	Bedienungsanleitung mobile Terminals	*A5_16 Bedienungsanleitung mobile Terminals.pdf*
A5.17	Softwareprojekt der Android-App	*A5_17 Probenmonitoring_App.zip*
A5.18	Datenblatt ZwickiLine Universalprüfmaschine	*A5_18 Material_Pruefmaschine_zwickiLine_Z0_5_bis_Z5_0_PI_DE.pdf*

A5.19	Datenblatt Kaptur KP1401 Tischscanner	*A5_19 KAPTUR_KP1401_DB-DE_Datenblatt.pdf*
A5.20	Bedienungsanleitung Zwick to Detact Import Tool	*A5_20 Bedienungsanleitung Zwick to Detact Import Tool.pdf*
A5.21	Inno Setup Script des Zwick to Detact Import Tools	*A5_21 Zwick_to_Detact_Import_Tool_setup.iss*
A5.22	Softwareprojekt des Zwick to Detact Import Tools	*A5_22 Zwick_to_DETACT.zip*
A5.23	Inno Setup Skript der ID Vergabe Software	*A5 23 ID_Vergabe_setup.iss*
A5.24	Softwareprojekt der ID Vergabe Software	*A5_24 ID_Generator.zip*
A5.25	Bedienungsanleitung ID Vergabe Software	*A5_25 Bedienungsanleitung ID Vergabe Software.pdf*
A6.1	Detact REST Schnittstelle	*A6_1 Detact REST API Dokumentation.pdf*
A6.2	Versuchsdaten als Origin Projekt	*A6_2 Umgebungsmonitoring-Data.opju*
A6.3	Pointreducer Python Script	*A6_3 pointreducer.py*
A6.4	Rohdaten Zugversuche	*A6_4 Rohdaten Zugversuche.zip*