FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Prozessüberwachung mithilfe von IoT

Prozessdiagnose einer Waschmaschine

Studiengang: Industrial Engineering
Modul: Datenmanagement, Leittechnik und statistische Prozesslenkung

Jens Juraschek	3291154	jens.juraschek@alumni.fh-aachen.de
David Wergen	3115266	david.wergen@alumni.fh-aachen.de

Inhaltsverzeichnis

Abbildungsverzeichnis

Prozessdiagnose einer Waschmaschine

Problemstellung

Jedes technische Gerät durchlebt einen jeweils individuellen Prozessablauf. Wie häufig dieser stattfindet, ist gerätespezifisch. Speziell im industriellen Umfeld und der dort etablierten Prozessleittechnik ist eine permanente Prozessüberwachung und Klassifikation von Betriebszuständigen von hohem Interesse. Kontinuierlich werden Prozessdaten erhoben, übermittelt und ausgewertet. Dadurch können Aussagen über den aktuellen Status der Maschinen getroffen und Standzeiten, Wartungsintervalle und weitere Betriebskennzahlen statistisch prognostiziert werden. Neben der technischen Umsetzung der Datenerfassung und -übertragung nimmt das Datenmanagement bei diesem Vorhaben eine tragende Rolle ein.

Im Rahmen dieser Projektarbeit ist der Prozess einer handelsüblichen Waschmaschine zu untersuchen. Auf dieser Problemstellung basierend stehen folgende Kernaufgaben im Fokus:

- Erfassung signifikanter Vibrationen
- Datenmanagement und Datenklassifikation
- Diagnose der Betriebszustände

Ziele

Das grundsätzliche Ziel der Prozessüberwachung und -diagnose einer Waschmaschine im Zuge dieser Projektarbeit ist es, alle Betriebszustände entlang des Prozessablaufs zu identifizieren und visualisieren. Daneben hat sowohl eine klare Lösungsstruktur als auch ein angemessenes Datenmanagement hohe Priorität.
Das Fundament für eine ingenieurwissenschaftliche Prozessüberwachung bildet ein stabiler Datenfluss. Dieser stützt sich auf optimierte Kommunikationsschnittstellen, sowie die Erfassung von adäquaten Daten.

Prozessbeschreibung und Ressourcen

Struktur zur Zielerreichung

In Abbildung 1 ist die Struktur der zur Zielerreichung gewählten Lösungsmethodik dargestellt.

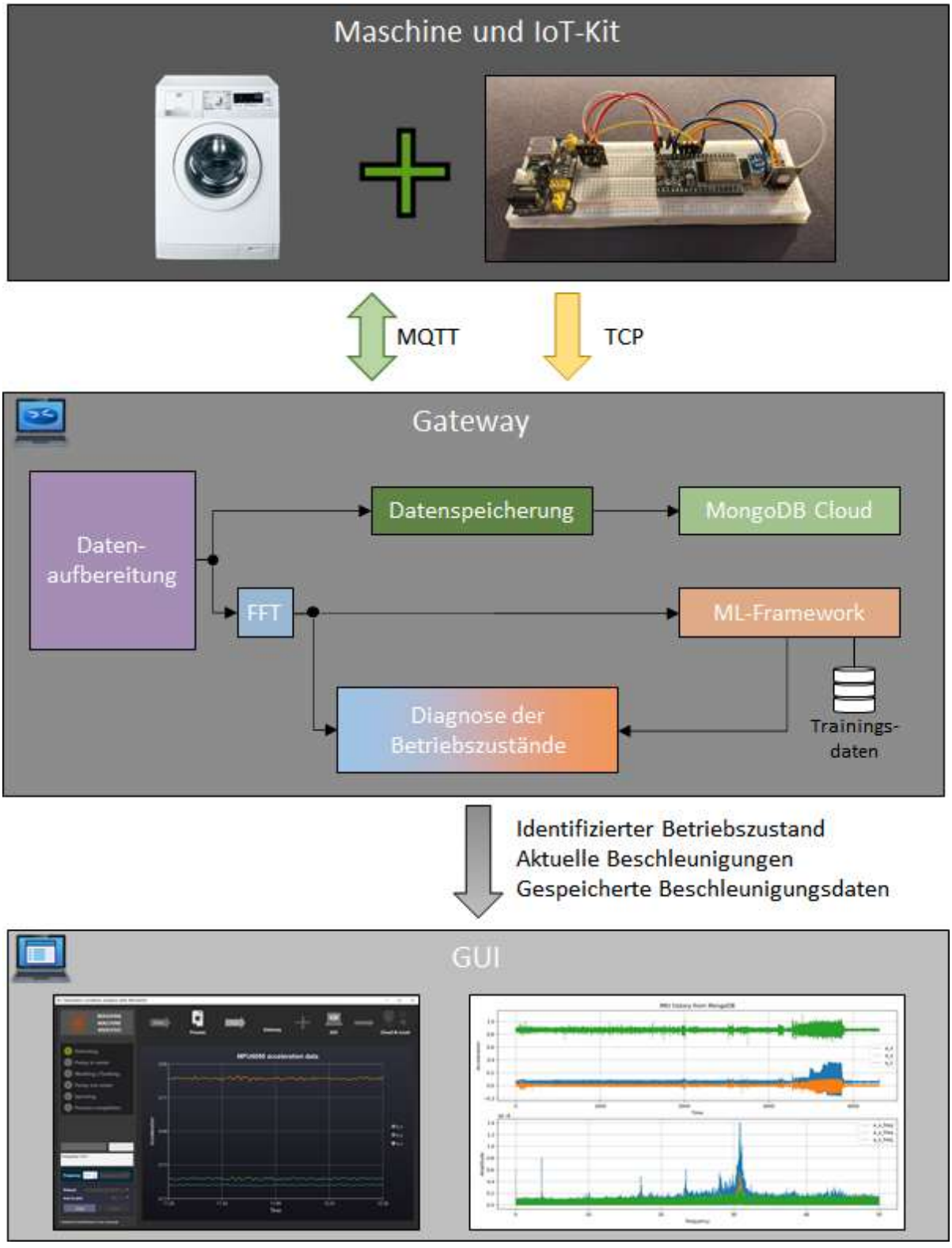

Abbildung 1: Struktur der Lösungsmethodik zur Zielerreichung

Am Ort des Geschehens befindet sich die zu untersuchende Waschmaschine und ein zusammengestelltes IoT-Kit. Dem auf dem IoT-Kit befindlichen Microcontroller wird vor der Datenaufnahme die geforderte Abtastfrequenz via MQTT mitgeteilt. Die Aufgabe des IoT-Kits ist es, Beschleunigungsdaten zu erfassen und diese mittels stabiler TCP-Verbindung als Bytes codiert vollständig an das Gateway zu übermitteln.
Als Gateway für diese Projektarbeit wird zwecks Mobilität ein Laptop verwendet. Dort werden die vom Microcontroller gesendeten Beschleunigungsdaten empfangen. Zur weiteren Verarbeitung werden die gesendeten Bytes decodiert und datentechnisch aufbereitet. In aufbereiteter Form werden diese Daten anschließend in einer MongoDB Atlas Datenbank gespeichert. Parallel findet eine Fourier-Frequenz-Transformation der erhaltenen Beschleunigungsdaten statt. Auf Basis des spezifischen Frequenzspektrums einzelner Datensätze ist mithilfe des maschinellen Lernens und der Datenklassifikation eine Identifikation des aktuellen Betriebsvorgangs möglich. Ein im Vorfeld aufgenommener und klassifizierter Trainingsdatensatz bildet die Grundlage für das maschinelle Lernen.
Die aufgenommenen Beschleunigungsdaten, der identifizierte Betriebszustand und Daten aus der MongoDB-Historie werden abschließend der GUI zur Visualisierung bereitgestellt. Die Beschleunigungsdaten des aktuellen Durchlaufs werden in einem Diagramm abgebildet, während eine Visualisierung des Betriebszustandes mittels Leuchten-System stattfindet. Findet keine Untersuchung statt oder wurde eine Untersuchung abgeschlossen, können via Dropdown Menü, die Beschleunigungsdaten und der gesamte Frequenzbereich, sowohl des aktuellen als auch der historischen Untersuchungsdurchläufe vollständig abgebildet werden.

Verwendete Hardware

Abbildung 2 zeigt die für das IoT-Kit verwendete Hardware.

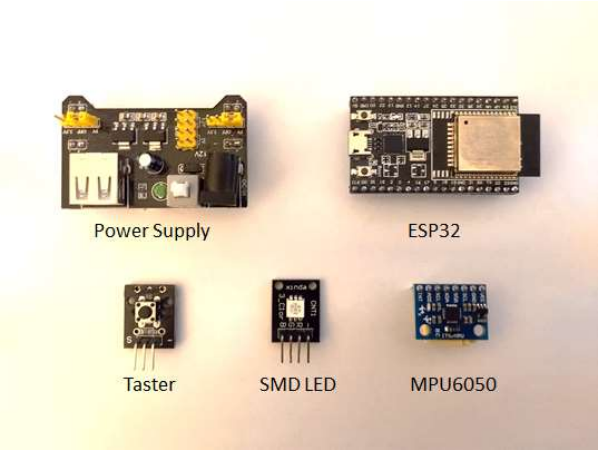

Abbildung 2: Verwendete Komponenten

Mithilfe des Power Supplys wird der ESP32 mit Elektrizität versorgt. Der Microcontroller ist die zentrale Einheit zur Kommunikation der vom MPU6050-Sensor aufgenommenen

Beschleunigungsdaten. Ein SMG RGB LED Modul teilt dem Anwender den aktuellen Status des IoT-Kits mit. Bei einem WiFi Verbindungsaufbau leuchtet das Modul Blau, im Zuge der MQTT Kommunikation gelb und während der gesamten TCP-Verbindung grün. Fehlgeschlagene Verbindungsversuche werden durch rotes Blinken signalisiert. Das Beenden eines Untersuchungsdurchlaufs wird durch Drücken des Tasters eingeleitet.
In Abbildung 3 ist das vollständig zusammengestellte IoT-Kit zu sehen. Von links ausgehend der Power Supply, der Taster, der ESP32 Microcontroller, der MPU6050 Sensor sowie die LED.

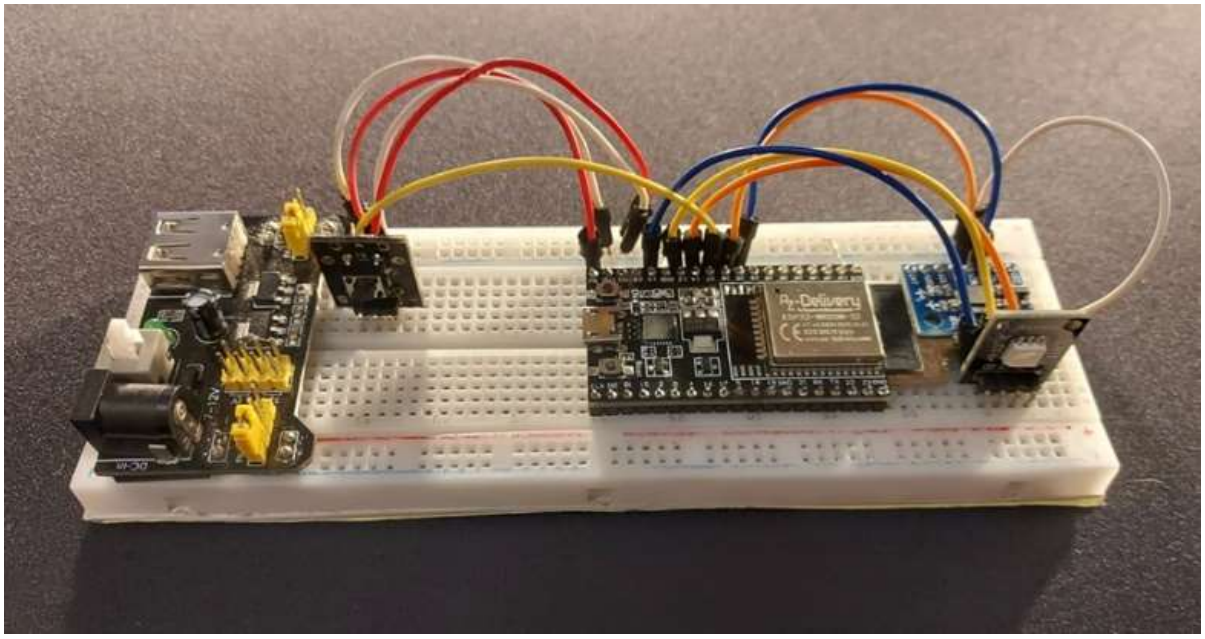

Abbildung 3: Komponenten als IoT-Kit

Verwendete Software

Um den Microcontroller ESP32 mit Leben zu füllen, muss eine geeignete Entwicklungsumgebung zur Programmierung verwendeten werden. Im Rahmen dieser Projektarbeit wird die Software „Thonny 3.3.13“ verwendet.

Neben dem Programm für den Microcontroller spielt das Gateway eine wichtige Rolle. Für dessen Programm wird die Entwicklungsumgebung „PyCharm Community Edition 2021.3“ verwendet.

Aus PyCharm heraus wird die Benutzeroberfläche initialisiert und geladen. Mithilfe des „Qt-Designer“ von Pyside6 wird diese erstellt und dem Gateway-Programm als Import zur Verfügung gestellt.

Programmablaufplan des Gateways

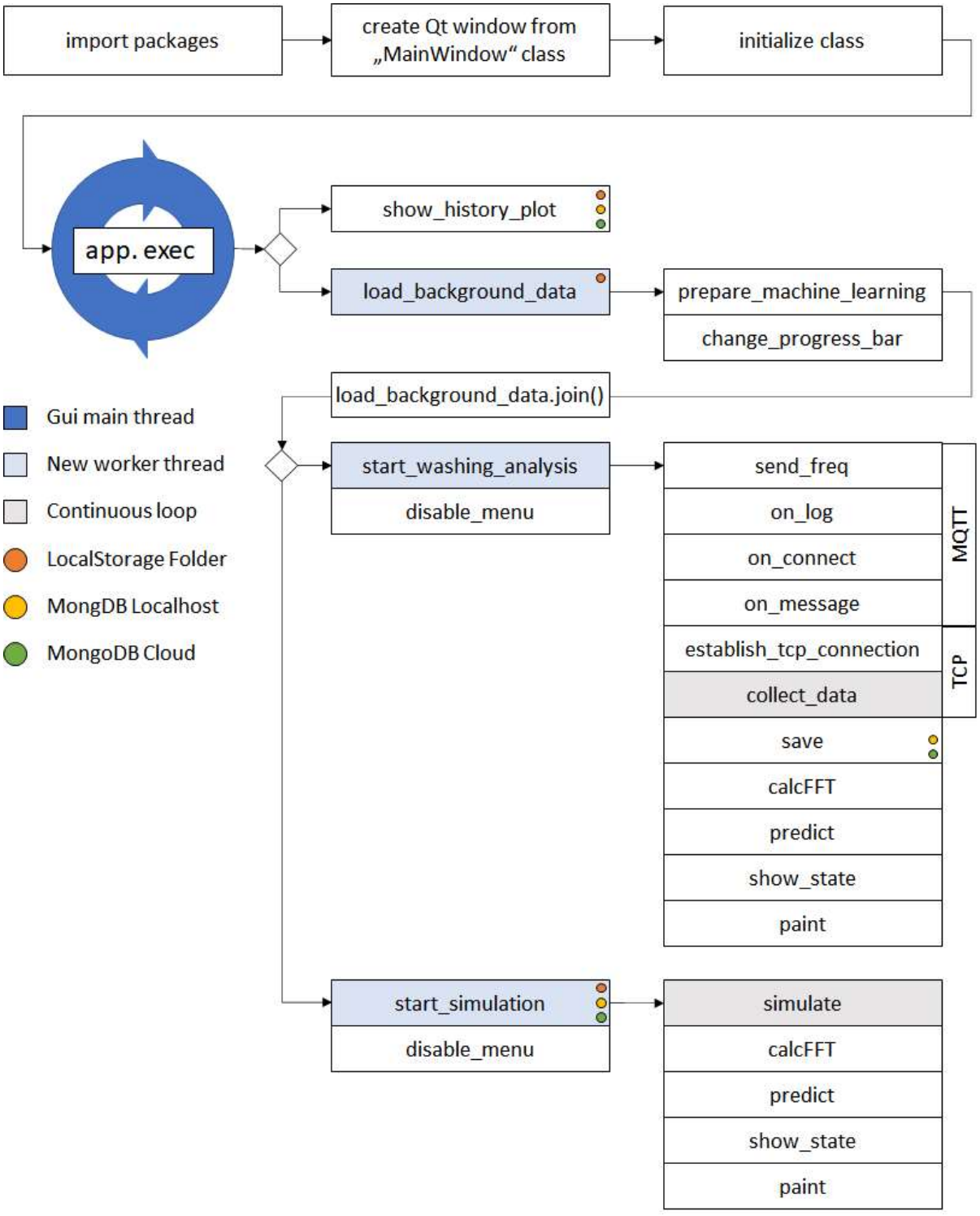

Abbildung 4: Programmablaufplan des Gateway-Programms

Im Gateway kann über ein GUI die Prozessaufnahme, -simulation und die Resultatanzeige historischer Aufnahmen durchgeführt werden. Bei der Prozessaufnahme erhält das Gateway kontinuierlich Daten vom Microcontroller. Die Visualisierung, das Speichern und die Betriebszustandsermittlung finden batchweise rund alle 100 Datensätze statt.

Programmablaufplan des Microcontrollers ESP32

Der Microcontroller benötigt zur Zielerreichung ebenfalls ein passendes Programm. Es ist in Micropython programmiert und der grundlegende Programmablauf nach dem Boot-Vorgang ist in Abbildung 5 dargestellt.

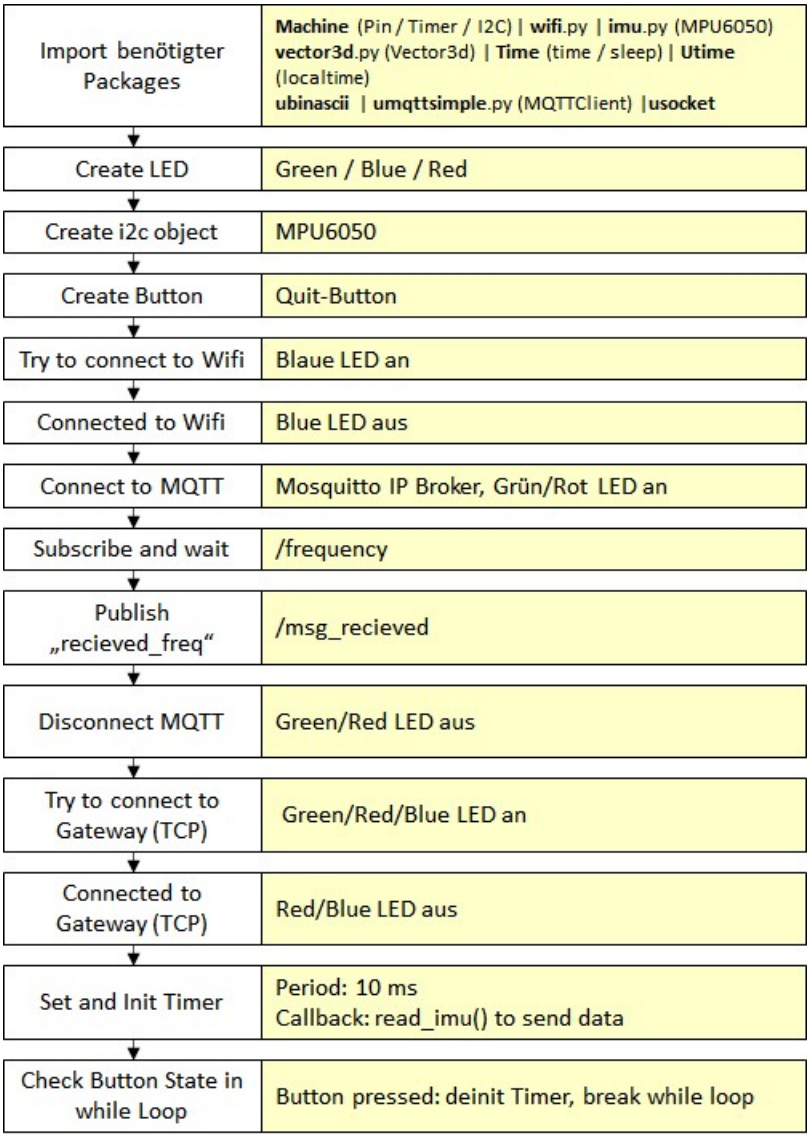

Abbildung 5: ESP32 Programmablaufplan

Alle 10 ms löst ein Timer die Callback-Funktion „read_imu“ aus. In dieser Funktion fragt der Microcontroller die vom Sensor aufgenommenen Beschleunigungen der drei Raumachsen und den Aufnahmezeitpunkt ab. Die Beschleunigungswerte werden auf fünf Nachkommastellen reduziert und zusammen mit dem Aufnahmezeitpunkt in einen String überführt. Hierbei trennt ein Absatzzeichen „\n“ den Zeitpunkt von den Werten und ein weiteres die Datensätze untereinander. Die Beschleunigungswerte selbst werden durch Kommata eindeutig getrennt. Nach Abschluss einer Datensatzaufnahme und -aufbereitung wird der String in Bytes encodiert und mittels einer TCP-IP Verbindung zum Gateway gesendet.

Herausforderungen der Problemstellung

Frequenzübermittlung vom Gateway zum Microcontroller

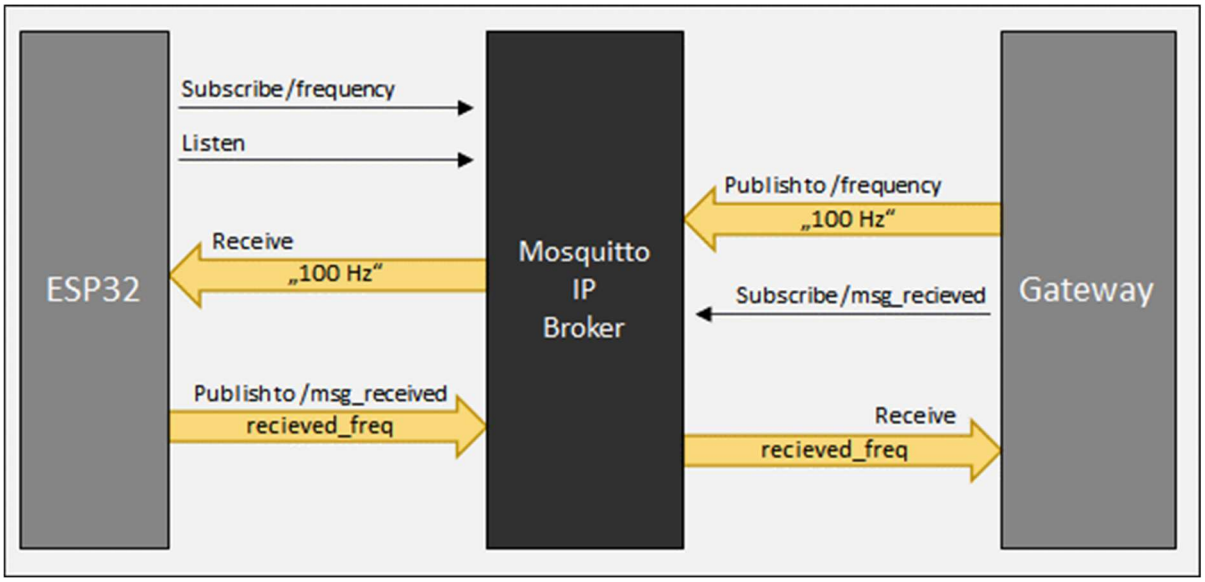

Abbildung 6: Übertragung der gewünschten Abtastrate von 100 Hz via MQTT

Die Frequenzübermittlung findet mittels MQTT statt. Hierzu wird ein Mosquitto IP Broker eingerichtet. Die Kommunikation zwischen Gateway und Microcontroller findet gemäß Abbildung 6 statt.

Datenverluste und effektive Abtastrate

Bei den vorherrschenden WiFi-/Bandbreitenkonditionen, der programmbedingten Abtastleistung und der Datenaufbereitung des Microcontrollers kommt es im Zuge der Datenübertragung mittels TCP-IP zu Datenverlusten. Aus einer Vielzahl von Testläufen resultiert eine übertragungsbedingte Reduktion der eingestellten Abtastfrequenz um 29%. Um das Risiko des Aliasing Effekts bei der Datenaufnahme so gering wie möglich zu halten, sollte die Abtastfrequenz mindestens dem doppelten Wert der aufzunehmenden maximalen Signalfrequenz entsprechen. Die maximale Waschmaschinendrehzahl von 1400 min^{-1} entspricht einer Signalfrequenz von 23,33 Hz. Die zur Prozessaufnahme verwendete Abtastfrequenz sollte demnach einer Mindestfrequenz von 46,67 Hz entsprechen.
Unter den vorliegenden Rahmenbedingungen wird eine Abtastfrequenz von 100 Hz als Standard eingestellt. Diese wird für die Erzeugung des Trainingsdatensatzes für den Classifier und zur Verifikation der Betriebszustandsvorhersage verwendet.
Bei der Erstellung des Trainingsdatensatzes sind vom Microcontroller während einer Laufzeit von 4.306 Sekunden und einer Abtastrate von 100 Hz 430.600 Datensätze gesendet worden. Das Gateway hat 307.903 Datensätze empfangen und verarbeitet. Dies entspricht einem

Datenverlust von 28,49 %. Die effektive Abtastfrequenz liegt mit 71,5 Hz über der Mindestfrequenz und das Risiko des Aliasing Effekts ist minimiert.
Mit der Erhöhung der Abtastfrequenz über 100 Hz geht ein Datenverlust in äquivalenter Höhe einher, sodass sich keine Verbesserung der effektiven Abtastfrequenz einstellt.

Anbindung und Gestaltung der GUI

Aus Abbildung 4 ist ersichtlich, dass am Gateway das Multithreading zum Einsatz kommt. Dies ist an dieser Stelle aufgrund der blockierenden Eigenschaft der TCP-IP Verbindung notwendig. Der Aufbau der TCP Verbindung funktioniert mithilfe eines Sockets. Um einen ungestörten Datentransfer zu gewährleisten, wird die restliche Ausführung des Programmcodes bei der Bindung des Socket blockiert. Dies führt bei einem zeitaufwendigeren Datenaustausch zum Einfrieren der GUI, sodass die weitere Bedienung oder programmtechnische Ansteuerung nicht möglich ist.
Um dieses Problem zu umgehen, wird die GUI als Main-Thread gestartet und weitere Worker-Threads zur Aufgabenbewältigung gestartet. Für das Trainieren des Classifiers, die Prozessaufnahme und die Prozesssimulation wird jeweils ein eigener Worker-Thread gestartet. Lediglich beim Ersteren wird zwecks Programmstabilität auf die Beendigung des Threads gewartet und das GUI blockiert.
Diese Art der Programmierung ermöglicht einerseits eine durchgängige und flüssige Bedienbarkeit, andererseits können Threads nur einmal gestartet werden. Da sie nicht gelöscht werden können, muss nach jeder der genannten Aktionen das Programm neugestartet werden. Eine weitere programmtechnische Optimierung würde an dieser Stelle allerdings den gesetzten Rahmen der Projektarbeit übersteigen.

Klassifikation der Trainingsdaten

Für die Betriebszustandsvorhersage ist ein Training des verwendeten Classifiers erforderlich. Hierzu dient der Beschleunigungs-Datensatz einer Prozessaufnahme als Grundlage. Die Aufbereitung der Rohdaten zum Training des Classifiers findet via Jupyter Notebook statt. Der Datensatz ist aus der MongoDB Cloud ins Notebook importiert worden. Weiterführend findet die Kategorisierung der Beschleunigungsdaten mithilfe einer VBA-basierten Excel-Datei statt. Die resultierenden Kategorien und dazugehörigen Zeitabschnittszuordnungen werden anschließend ins Jupyter Notebook kopiert. Beim Training des Classifiers ist das Format der vorherzusagenden Datensätze zu beachten. Bei der Prozessaufnahme soll der Betriebszustand auf Basis von 100 absoluten Geschwindigkeitswerten und dessen Frequenzspektrum identifiziert werden. Daher muss der Classifier mit einem Amplituden-Dataframe trainiert werden, welches 51 Spalten (50 Amplituden, 1 Kategorie) und mindestens

zwei Layer pro Kategorie aufweist. Die Amplitudenberechnung erfolgt hierbei ebenfalls batchweise mit jeweils 100 Beschleunigungsdatensätzen, sodass 50 Amplitudenwerte je Layer resultieren.

Symmetriebedingt ist es erforderlich, dass für jede Kategorie die gleiche Anzahl an Datensätzen verarbeitet werden. Da die Kategorien unterschiedlich viele Datensätze beinhalten können, wird die maximal zu verwendende Datensatzgröße auf die Größe des Datensatzes mit der geringsten Anzahl an Einträgen beschränkt.

Um gleichzeitig die Vorhersagegenauigkeit des Classifiers angemessen hochzuhalten, ist es notwendig die Trainingsdaten durch weitere Prozessaufnahmen zu erweitern. Auf diese Art und Weise steigt der Inhalt der Kategorie mit der geringsten Anzahl an Datensätzen. Mit jeder Prozessaufnahme geht eine zeitaufwendige manuelle Klassifizierung des Ausgangssignals mit etwaigen Ungenauigkeiten einher. Eine entsprechende Vervielfältigung des bereits klassifizierten Datensatzes führt unter Reduzierung der manuellen Klassifizierungsungenauigkeit und einer Zeitersparnis ebenfalls zum gewünschten Ergebnis.

Abschließend wird der erzeugte Amplituden-Dataframe als CSV-Datei lokal abgespeichert und dem Gateway zum Classifier Training bereitgestellt.

Informationen zum Testbetrieb

Schritte zur Inbetriebnahme des IoT-Kit mit LEDs und Taster

1. WLAN in „wifi.py“ einrichten, IP-Adresse in main.py Zeile 18 ändern und Ordnerinhalt von ESP32 (ohne „main_default.py“) flashen.
2. IP-Adresse in gateway.py Zeile 31 ändern.
3. MongoDB Cloud Adresse in gateway.py Zeile 32 ändern.
4. MQTT Mosquitto IP Broker starten. (Einrichtung: How To Setup An MQTT System (Broker & Client) | The Automation Blog)
5. Gateway starten.
6. ESP Microcontroller starten.
7. Bei positiver Rückmeldung des ESPs (orangene LED) auf „Connect to ESP“ im Gateway klicken. Wenn möglich wird nun per MQTT die eingestellte Frequenz zum ESP gesendet (grüne LED) und eine TCP IP-Verbindung (weiße LED) zum Datentransfer zwischen Gateway und ESP errichtet. Der aktive Datentransfer wird durch die konstant grün leuchtende LED signalisiert.
8. Nach Abschluss der Aufnahme muss der Taster am IoT-Kit gedrückt werden.
   Bei Verwendung einer IDLE werden die Laufzeit, die Frequenz und die Soll-Anzahl der zu sendenden Daten ausgegeben. Das Gateway schaltet daraufhin automatisch die Datenaufnahme und -verarbeitung ab.

Schritte zur Inbetriebnahme des IoT-Kit ohne LEDs/Taster

1. WLAN in „wifi.py“ einrichten, IP-Adresse in main_default.py Zeile 18 ändern und Ordnerinhalt von ESP32 (ohne „main.py“) flashen („main_default.py“ auf dem ESP als „main.py“ abspeichern!).
2. IP-Adresse in gateway.py Zeile 31 ändern.
3. MongoDB Cloud Adresse in gateway.py Zeile 32 ändern.
4. Gateway starten.
5. Klick auf „Connect to ESP“, warten bis das Programm „Listening…“ zurückmeldet.
6. Main.py auf dem ESP Microcontroller starten. Die Abtastfrequenz ist konstant auf 100 Hz eingestellt.
7. Da die Aufnahme ohne LEDs und Taster lediglich dem Funktionsnachweis dient, wird der Aufnahmeprozess nach einer Zeit von 100 s automatisch beendet. Bei Verwendung einer IDE werden die Laufzeit, die Frequenz und die Soll-Anzahl der zu sendenden Daten ausgegeben. Das Gateway schaltet daraufhin automatisch die Datenaufnahme und -verarbeitung ab.

Benötigte Bibliotheken

```
import sys
import os
import threading
import Qt.gui
import Qt.icons.icons
from Qt.gui import *
import PySide6.QtCharts
from PySide6 import QtCharts
import usocket as socket
import paho.mqtt.client as mqtt
from pymongo import MongoClient, errors
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from time import localtime, sleep
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
```

Kein MQTT IP Broker zur Frequenzübermittlung

Sollte keine Einrichtung eines Mosquitto IP Brokers stattfinden, kann zur Frequenzübertragung auf einen Testserver ausgewichen oder die „main_default.py“ auf dem ESP verwendet werden. Letztere Variante liefert eine Abtastfrequenz von 100 Hz.

Kein Zugriff auf die MongoDB Cloud

Um Zugriff auf die verwendete MongoDB Cloud zu erhalten, muss die aktuelle IP-Adresse des Anwenders in MongoDB Atlas hinterlegt werden. Das Programm ist so konzipiert, dass es bei fehlender Cloud-Anbindung aufgenommene Daten automatisch lokal auf dem Rechner via MongoDB-Localhost abspeichert.

Datendownload für Plot oder Simulation

Bei beiden Varianten der Datenverarbeitung probiert das Programm eine festgelegte Reihenfolge von Downloadmöglichkeiten aus:

1. Ordner LocalStorage/MongoDB_collections
2. MongoDB Atlas Cloud
3. MongoDB Localhost

Auf diese Art und Weise wird die Bereitstellung der Daten gewährleistet und die Programmperformance erhöht.

Performancesteigerung der „read_imu“-Funktion

In der „vector3d.py“-Datei wurde folgende Optimierung vorgnommen:

```
@property
def xyz(self):
    self.update()

    return '{0:3.5f},{1:3.5f},{2:3.5f}\n'.format(self._calvector[self._transpose[0]] * self._scale[0],
            self._calvector[self._transpose[1]] * self._scale[1],
            self._calvector[self._transpose[2]] * self._scale[2])
```

Abbildung 7: Optimierung der "vector3d.py"-Datei

Die Abfrage „accel.xyz“ in der „read_imu“-Funktion liefert statt eines Tuples einen formatierten String der Beschleunigungswerte. Auf diese Art und Weise entfällt das Verarbeiten oder Eliminieren der runden Klammern und die Beschleunigungswerte müssen nicht einzeln abgefragt werden.