FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Fachhochschule Aachen

Fachbereich Maschinenbau und Mechatronik

Projektbericht

Prozessüberwachung und Zustandsklassifikation einer Waschmaschine

Wintersemester 2024/2025

Modul 81307 – Datenmanagement und Leittechnik

vorgelegt von: Fatih Sözer - 3245736

Sebastian Jessen - 3271178

Referent: Prof. Dr. Ing. Stephan Kallweit

Prof. Dr. Ing. Adam Fiolka

Abgabedatum: 05.03.2025

Inhaltsverzeichnis

1 Aufbau des Projektes

In diesem Projekt werden die Schwingungen einer Waschmaschine mittels einer Inertial Measurement Unit (IMU) erfasst und über einen D1Mini übertragen. Ziel ist die Klassifikation von vier Betriebszuständen der Waschmaschine anhand der erfassten Daten. Der gesamte Aufbau ist in Abbildung 1 dargestellt. Zur Gewährleistung einer skalierbaren und strukturierten Codebasis wurde ein objektorientierter Ansatz in Python gewählt. Dies ermöglicht eine flexible Erweiterung und effiziente Umsetzung für komplexere Aufgabenstellungen.

Datenaufnahme und Versandung

- **Hardware**: ESP8266 Mikrocontroller mit WLAN-Funktion und MPU6050 (IMU), verbunden über I2C Schnittstelle
- Hardware an Seite der Waschmaschine mit Doppelseitigem Klebestreifen geklebt, um Vibrationen und Schwingungen über die Blechwand besser zu übertragen
- Erdbeschleunigung wirkt hierbei auf die x-Achse
- D1Mini dient als TCP-Client

Daten- und Benutzerschnittstelle

- **Hardware**: Macbook, welcher als TCP-Server dient
- Visual Studio Code (VSC): Code-Editor, zum Programmieren genutzt mit Erweiterung PyMakr
- QtCreator: Entwerfung grafische Benutzeroberfläche (GUI)

Daten- und Benutzerschnittstelle

- MongoDB Atlas: Cloudbasierte NoSQL Datenbank
- MongoDB Compass: Benutzerfreundliche Verwaltung der lokalen und cloudbasierten Daten
- Zum Erhöhen der Verfügbarkeit lokal und cloudbasierte Speicherung
- Abrufen der Daten durch Simulation oder Datenverarbeitung in Jupyter-Notebook

Datenverarbeitung

- Jupyter-Notebook: Vergleich und Auswahl der ML-Modelle und abspeichern für Vorhersagen

Für genauere Funktion der Skripte siehe Anhang A: Skriptbeschreibungen.

Git: https://github.com/Crank37/DLSP; https://git.fh-aachen.de/fs1761s/dlsp-projekt

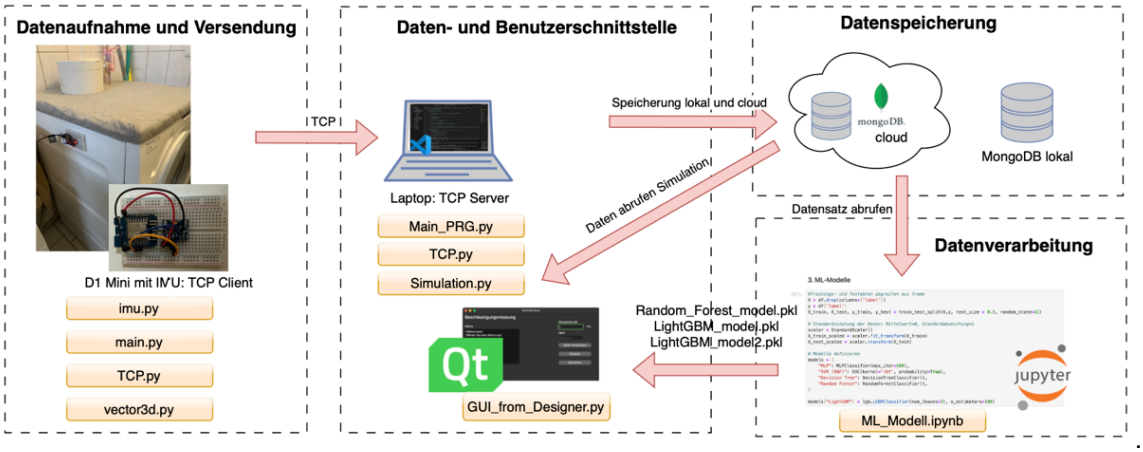


Abbildung 1: Aufbau Toolchain und Versuch

2 Datenaustausch

Zur effizienteren Verarbeitung der Daten und Reduzierung Datenmenge wurde der Ansatz gewählt, Sensordaten gebündelt in Datenpaketen zu übertragen. Dadurch wird vermieden, die Daten kontinuierlich während des gesamten Waschvorgangs zu erfassen und einen Zeitstempel zu vergeben. Jedes Paket wird mit einem Label versehen.

Datenerfassung
Die dominierende Zeitkonstante wurde basierend auf der maximalen Schleuderdrehzahl von 1200 U/min = 20 U/s angenommen, was 20 Hz entspricht. Entsprechend dem Nyquist-Kriterium gilt:

$$f_{abt} \geq 2 \cdot f_{dom} = 2 \cdot 20Hz = 40Hz$$

Im Projekt wurden die Abtastraten für die Beschleunigungen ax, ay und az sicherheitshalber auf 100 Hz gesetzt. Im Programm werden die Beschleunigungswerte in Listen in jeweils 128 oder 256 Samples (auswählbar) zusammengetragen. Um eine genauere und verlässlichere Klassifikation durchzuführen, wurden die 256 Samples bevorzugt. Für den Trainingsdatensatz wird das korrekte Label während der Datenerfassung der Waschmaschine durch direkte Prozessbeobachtung gesetzt.

Datengröße
Die Datenübertragung erfolgt über TCP, um eine zuverlässige Übertragung der Datenpakete zu gewährleisten. Im Gegensatz zum UDP-Kommunikationsprotokoll, das für eine schnelle Übertragung großer Datenmenge zuständig ist, liegt der Fokus hier auf Datenintegrität und der korrekten Reihenfolge der Pakete. Alle Daten müssen vollständig und ohne Verluste empfangen werden. Die Daten werden ca. alle 2,56 s als Batch versendet.

TCP unterstützt beim einmaligen Versenden 1500 Byte. Um sicherzustellen, dass die zu sendenden Datenpakete die Grenze von 1500 Byte nicht überschreiten, werden diese in sogenannten „chunks“ (Teilpakete) zerlegt. Die Werte werden als *FLOAT* gespeichert und mithilfe der *struct*-Bibliothek Byte-kodiert. Für die Speichergröße der drei Beschleunigungen mit jeweils 256 Samples und vier Teilpaketen gilt:

$$Speicher = \frac{3 \cdot Floats \cdot Samplegröße}{Teilfaktor} = \frac{3 \cdot 4Byte \cdot 256}{4} = 768\ Byte$$

Die Aufnahme der Beschleunigungswerte, sowie die Übertragung der Daten an den TCP **Server** geschieht durch die Methode *data_struct_send()* aus der Klasse *TCPClient* über den **Client**. Empfangen werden die Daten mit *receivedata()* aus der Klasse *TCPServerMDB.*

Speicherung
Zur schnellen Verarbeitung unstrukturierter und großer Datenmengen (Prozessdaten) wird das DBMS MongoDB verwendet. Nach dem Empfang der Daten vom TCP Client werden diese über die Methode *getindb()* innerhalb der Klasse *TCPServer_MDB* in die Datenbank eingetragen (s. gespeicherte Objektstruktur in Abbildung 2). Zusätzlich sind die Datenbankinstanzen im Anhang E: Datenbankinstanzen dargestellt.

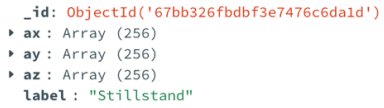


Abbildung 2: gespeichertes Objekt in MongoDB

Mit *create_opendb()* wird eine Verbindung mit der lokalen und cloudbasierten Datenbank hergestellt sowie eine Datenbankinstanz erstellt. Der Zugriff auf den Cloud-Cluster ist über eine (für alle) freigeschaltete IP von jedem Standort aus möglich. Die Authentifizierung erfolgt über einen im Code hinterlegten Benutzernamen und ein Passwort, wodurch der Zugriff gewährt wird.

3 Machine Learning

Aus der Waschmaschine werden folgende Zustände klassifiziert:

Zustände	Beschreibung	Kategorisiertes Label (num.)
Stillstand	Aus	0
Schleudern	Schleudervorgang bis 1200 U/min	1
Waschen	Einweichen (Langsames Drehen)	2
Spülen	Schmutzwasser abpumpen und neues Wasser zufügen zum Abspülen	3

Die Datenvorverarbeitung, das Trainieren sowie das Speichern des Modells erfolgen in der Datei. *ML_Modell.ipynb* (Für andere Validierungen in *Skripte/WeitereJupyterVal/x.ipynb*). Die folgenden Beschreibungen beziehen sich auf die Herleitung zur ersten Validierung.

Trainingsdaten

Die Zielvariable ist der Zustand der Waschmaschine, welcher von einem String in einen numerischen Wert umgewandelt wird. Alle aus der MongoDB Cloud abgerufenen Daten werden in ein JSON-Objekt konvertiert. Basierend auf dem Betrag der Beschleunigungssachsen wird für jedes Objekt die Frequenztransformation durchgeführt, um das Signal in seine Frequenzkomponente zu zerlegen. Somit werden zur Klassifikation 128 unabhängige Variablen (s. Abbildung 3) genutzt. Um bessere Ergebnisse zu erhalten, werden hohe Frequenzen (hier 40Hz) mit einem Tiefpassfilter entfernt. Die Werte werden in einen *pandas* DataFrame eingetragen. Die Modelle wurden mit ca. 200 Datensätzen trainiert, wobei 30 % der Daten als Testdaten verwendet wurden. Zur besseren Vergleichbarkeit der Variablen erfolgt eine Standardisierung, in der die Standardabweichungen eins und Mittelwerte null sind.

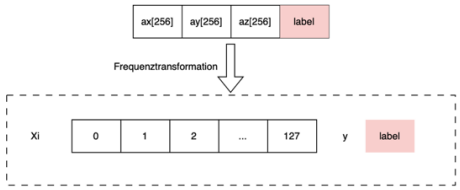


Abbildung 3: Erzeugung Trainingsdaten

ML-Modelle

Für die Klassifikation wurden verschiedene ML-Modelle aus der *scikit-learn* Bibliothek miteinander verglichen (s. Abbildung 4). Da das Modell der höchsten Klassifikationsgüte, bewertet anhand der Genauigkeit und Präzision, bei Random Forest liegt, wird dieses gewählt [92,5% Genauigkeit, 92,9% Präzision]. Nach der Auswahl des besten Modells wurde eine Cross-Validierung durchgeführt, um es weiter zu optimieren und die beste Wahl des Testdatensatzes zu ermitteln. Die Konfusionsmatrix (s. Abbildung 5) zeigt die richtig oder falsch vorhergesagten Zustände aus dem Testdatensatz.

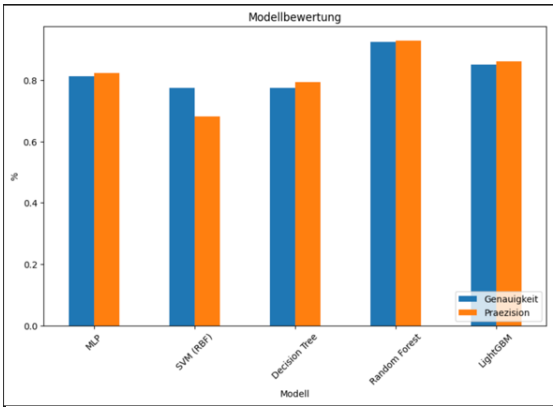


Abbildung 4: Bewertet Modelle zur Klassifikation

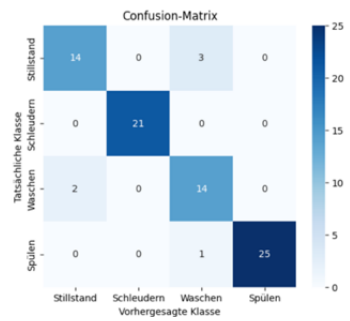


Abbildung 5: Confusion-Matrix

Von insgesamt 70 Datensätzen wurden fünf fehlerhaft zugeordnet, darunter insbesondere der *„Stillstand“*, der fälschlicherweise als *„Waschen“* erkannt wurde. Dies stellte ein zentrales Problem dar, da der Waschvorgang nur geringe Schwingungen erzeugt und daher besonders anfällig für Fehlklassifikation (Stillstand) war. Das endgültige Modell mit den ermittelten Gewichten wurde als *Random_Forest.pkl* mit der *Joblib-Bibliothek* gespeichert, um es für *Vorhersagen* in der Simulation zu verwenden.

4 Anwendung und Fazit

Benutzeroberfläche

Die aus *Benutzeroberflaeche.ui* in QT Creator erstellte GUI wird in ein Python-Skript konvertiert und im *Main_PRG.py* eingebunden. Durch den Einsatz von *Threading* wird eine parallele Verarbeitung ermöglicht, sodass die GUI jederzeit bedienbar bleibt, während das Programm im Hintergrund weiterläuft (bspw. zum Stoppen). Die GUI (s. Abbildung 6) ist darauf ausgelegt, den Programmablauf für die Datenerfassung mit dem D1mini, sowie zur Durchführung der Simulation zu steuern. Zum Starten eines Vorgangs stehen die Buttons unten rechts *„Simulation“* und *„Daten empfangen“* zur Verfügung, während der Button *„Stoppen“* das laufende Programm unterbricht. Prozessparameter, wie das Abtastintervall für die Datenerfassung, können über das Texteingabefeld angepasst werden. Das Label-Feld ermöglicht die Zuordnung der Klasse während der Datenerfassung. Der Graph auf der linken Seite visualisiert das aktuelle Frequenzspektrum während der Simulation, während die Statusanzeige oben rechts den aktuellen Zustand der Klassifikation oder Datenübertragung ausgibt.

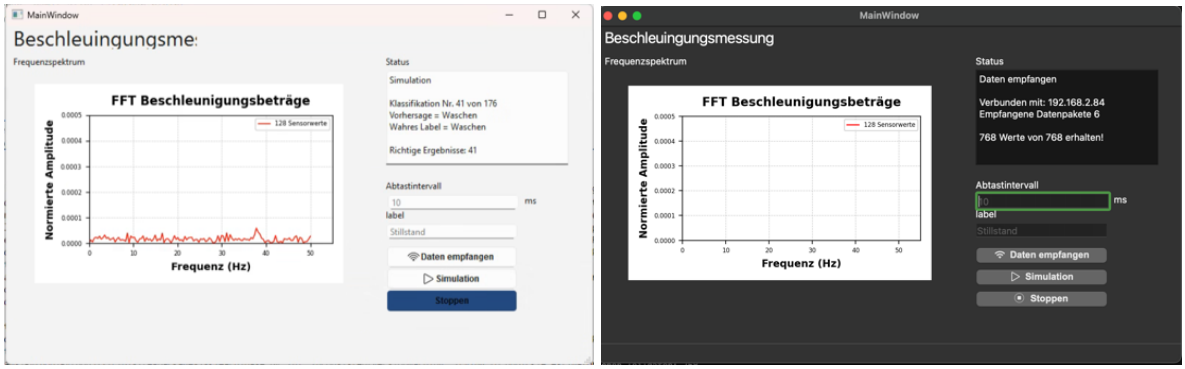


Abbildung 6: GUI links (Windows), rechts (MacOS, Darkmode)

Validierung

Um das Programm auszuführen, können die Schritte aus dem Anhang C: Inbetriebnahme befolgt werden. Hierbei sollten die Skripte aus Anhang A: Skriptbeschreibungen vorliegen.

Die Tests wurden sowohl unter macOS (Entwicklungsumgebung) als auch unter Windows durchgeführt. Bei Bibliotheksfehlern sollte ein Versionsabgleich (s. Anhang B: Installierte Pakete) erfolgen. Insbesondere wird empfohlen, *pymongo* in Version 4.10.1 zu verwenden, da neuere Versionen möglicherweise Probleme verursachen. Die Datenerfassung erfolgte bei einer zu ¾-gefüllten Waschmaschine, sodass die Schwingungen mehr hervorgehoben werden, mit einer Nenndrehzahl von 1200 U/min.

In der Statusanzeige der Benutzeroberfläche sind die Anzahl der erhaltenen Datenpakete, sowie erwartete und tatsächliche erhaltene Werte dargestellt (s. Abbildung 7). So kann identifiziert werden, ob alle Daten tatsächlich erhalten werden. Dies ist in den Tests immer der Fall gewesen, sodass 768 von 768 Werten erhalten wurden.

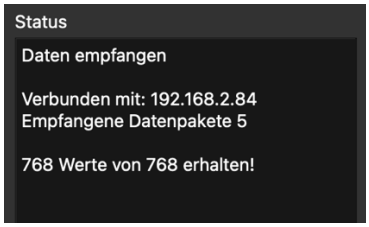


Abbildung 7: Ausschnitt aus GUI zur Sicherstellung Erhalt Datenpakete

Die Klassifikation innerhalb der Simulation wurde mit den Datensätzen aus *Messung27_02* (176 Datensätze) und *Messung23_02* (426 Datensätze) durchgeführt. Die Resultate sind in Abbildung 8 aufgeführt. Ersichtlich ist, dass der Zustand *„Stillstand“* größtenteils falsch interpretiert wird. In der Messung23_02 (zum Testen bei schlechten Datensätzen) sind nicht alle Datensätze 100 % genau gelabelt worden. Hierbei ist wieder zu sehen, dass größtenteils *„Stillstand“* mit dem Zustand *„Waschen“* falsch interpretiert wird. Diese und weitere Ergebnisse wurden übersichtlich im Anhang D: ML-Modelle Validierung dargestellt.

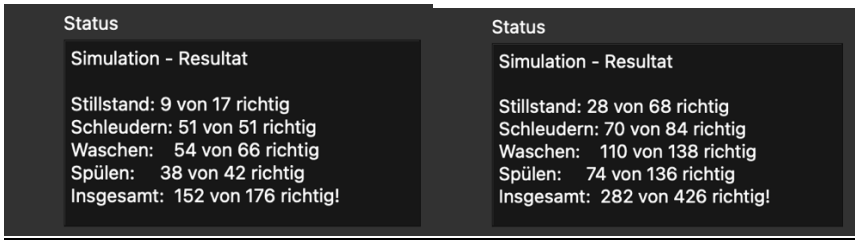


Abbildung 8: Resultate beider Datensätze im Status

Fazit

Zusammengefasst wurde in diesem Projekt erfolgreich eine Prozessüberwachung für eine Waschmaschine implementiert. Die benutzerfreundliche GUI ermöglicht die Steuerung, sowie Anzeigen der Klassifikationsergebnissen aus den Daten der Datenbank.

Auch wenn die Methode dieser Datenerfassung als sehr effizient für die Datenspeicherung und Live-Klassifikation erweist, zeigen sich jedoch Herausforderungen bei dem Labeln und der Datenaufnahme. Hierzu müssen über die GUI immer zu richtigen Zeiten die Daten aufgenommen werden, wogegen bei der kontinuierlichen Datenaufnahme die Bearbeitung im Nachhinein durchgeführt werden kann. Hierbei wiederum entsteht ein höherer Zeitaufwand für die nachträgliche Analyse und das Labeln.

Die Klassifikation erwies sich als herausfordernd, insbesondere aufgrund der starken Abdämpfung der Schwingung. Dies führte dazu, dass sich die Zustände *„Stillstand“* und *„Waschen“* nicht so einfach unterscheiden ließen. Dies sieht man sehr gut in der Validierung, in der die korrekte Identifizierung des Stillstands immer unter 40 % liegt. Hierzu könnten andere empfindlichere IMU-Sensoren verwendet werden, um eventuell das Problem zu beheben. Zudem waren die Spülvorgänge sehr kurz und gingen sehr schnell in das Spülschleudern über, sodass eine Verwechslung mit dem Schleudervorgang möglich war. Aus diesem Grund war die präzise Datenerfassung mit korrektem Label zum jeweiligen Zeitpunkt wichtig.

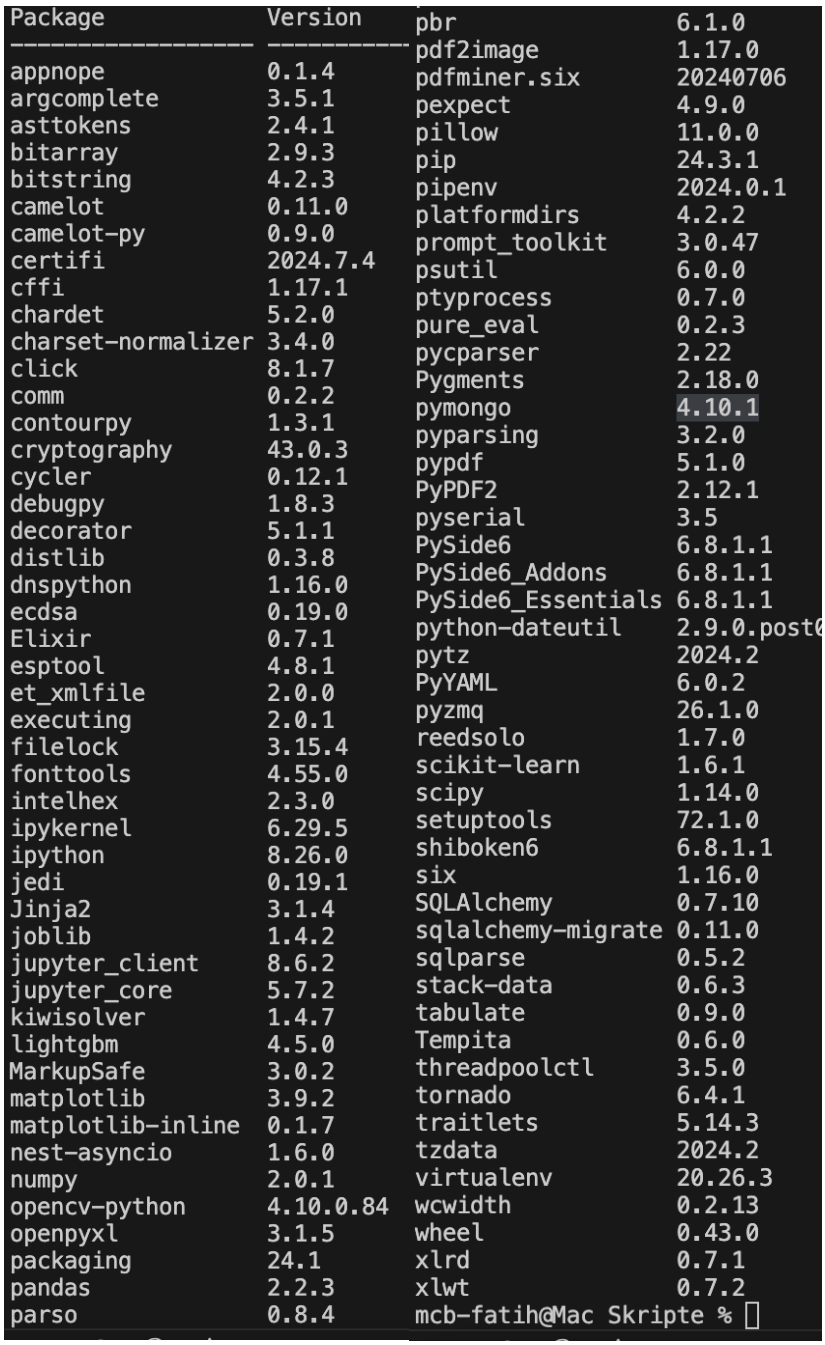
Um die Klassifikation breiter anwendbar zu machen, sollten künftig Daten mit unterschiedlichen Füllständen und Schleuderdrehzahlen erfasst werden. Das Modell erreichte eine solide Klassifikationsgenauigkeit von 92 %, wobei mögliche Anzeichen von Overfitting erkennbar sind. Die Übertragbarkeit auf andere Messergebnisse mit unterschiedlichen Füllständen, Drehzahlen oder aber auch andere Waschmaschinen könnte eingeschränkt werden. (s. Validierung)

Abbildungsverzeichnis

Anhang A: Skriptbeschreibungen

<table>
<tr><th>Hardware</th><th>Skript</th><th>Beschreibung</th></tr>
<tr><td rowspan="6">Lokal</td><td>Main_PRG.py</td><td>-Instanziierung der Klassen (Konfiguration Db, TCP)
-Verknüpfung Ui
<pre>class ServerController(
    host: str,
    port: int,
    cloud: int,
    dbcloudurl: str,
    dbinstance: str,
    MLModelname: str,
    simspeed: float
)</pre>-host: IP der TCP Verbindung
-port: Port der TCP Verbindung
-cloud: Abspeicherung Daten (0: lokal, 1:cloud, 2: beides)
-dbcloudurl: Serveradresse MongoDB Cloud
-dbinstance: Name für Erstellung/Zugriff der Datenbankinstanz (Simulation und Datenerfassung)
-MLModelname: Dateiname abgespeicherten ML-Modell
-simspeed: Simulationsgeschwindigkeit in s</td></tr>
<tr><td>TCP.py</td><td>-Klasse TCPServerMDB
-TCP Serverseite
-Empfang der Daten und hochladen in MongoDB</td></tr>
<tr><td>GUI_from_Designer.py</td><td>-Klasse Ui-MainWindow
-Erstellte Benutzeroberfläche mit QT Designer</td></tr>
<tr><td>ML_Modell.ipynb</td><td>-Skript zum Trainieren des Machine Learning-modells</td></tr>
<tr><td>Simulation.py</td><td>-Klasse Simulation</td></tr>
<tr><td>Random_Forest_model.pkl
LightGBM_modelx.pkl</td><td>-ML-Modell zur Vorhersage (gespeicherte Gewichtungen)</td></tr>
<tr><td rowspan="4">D1Mini</td><td>imu.py</td><td>-Klasse MPU6050</td></tr>
<tr><td>Vector3d.py</td><td>-Klasse Vector3d</td></tr>
<tr><td>Main.py</td><td>-Instanziierung der Klassen
<pre>class TCPClient(
    host: str = "127.0.0.1",
    port: int = 12345,
    ssid: str = "abc",
    pw: str = "abc",
    samplemode: int = 0
)</pre>-host, port: IP und Port für TCP-Verbindung
-ssid, pw: Netzwerk ssid, pw
-samplemode: 0 (128 Samples), 1 (256 Samples)</td></tr>
<tr><td>TCP.py</td><td>-Klasse TCPClient
-Datenerfassung
-Datenversendung an Server</td></tr>
</table>

Anhang B: Installierte Pakete



Package	Version
appnope	0.1.4
argcomplete	3.5.1
asttokens	2.4.1
bitarray	2.9.3
bitstring	4.2.3
camelot	0.11.0
camelot-py	0.9.0
certifi	2024.7.4
cffi	1.17.1
chardet	5.2.0
charset-normalizer	3.4.0
click	8.1.7
comm	0.2.2
contourpy	1.3.1
cryptography	43.0.3
cycler	0.12.1
debugpy	1.8.3
decorator	5.1.1
distlib	0.3.8
dnspython	1.16.0
ecdsa	0.19.0
Elixir	0.7.1
esptool	4.8.1
et_xmlfile	2.0.0
executing	2.0.1
filelock	3.15.4
fonttools	4.55.0
intelhex	2.3.0
ipykernel	6.29.5
ipython	8.26.0
jedi	0.19.1
Jinja2	3.1.4
joblib	1.4.2
jupyter_client	8.6.2
jupyter_core	5.7.2
kiwisolver	1.4.7
lightgbm	4.5.0
MarkupSafe	3.0.2
matplotlib	3.9.2
matplotlib-inline	0.1.7
nest-asyncio	1.6.0
numpy	2.0.1
opencv-python	4.10.0.84
openpyxl	3.1.5
packaging	24.1
pandas	2.2.3
parso	0.8.4
pbr	6.1.0
pdf2image	1.17.0
pdfminer.six	20240706
pexpect	4.9.0
pillow	11.0.0
pip	24.3.1
pipenv	2024.0.1
platformdirs	4.2.2
prompt_toolkit	3.0.47
psutil	6.0.0
ptyprocess	0.7.0
pure_eval	0.2.3
pycparser	2.22
Pygments	2.18.0
pymongo	4.10.1
pyparsing	3.2.0
pypdf	5.1.0
PyPDF2	2.12.1
pyserial	3.5
PySide6	6.8.1.1
PySide6_Addons	6.8.1.1
PySide6_Essentials	6.8.1.1
python-dateutil	2.9.0.post0
pytz	2024.2
PyYAML	6.0.2
pyzmq	26.1.0
reedsolo	1.7.0
scikit-learn	1.6.1
scipy	1.14.0
setuptools	72.1.0
shiboken6	6.8.1.1
six	1.16.0
SQLAlchemy	0.7.10
sqlalchemy-migrate	0.11.0
sqlparse	0.5.2
stack-data	0.6.3
tabulate	0.9.0
Tempita	0.6.0
threadpoolctl	3.5.0
tornado	6.4.1
traitlets	5.14.3
tzdata	2024.2
virtualenv	20.26.3
wcwidth	0.2.13
wheel	0.43.0
xlrd	0.7.1
xlwt	0.7.2

mcb-fatih@Mac Skripte %

Anhang C: Inbetriebnahme

Vorgang	*Schritte*
Skript starten	1. Öffnen von Visual Studio Code (VSC) oder vergleichbares 2. Ordner „Skripte“ über VSC öffnen 3. Python Skript Main_PRG.py ausführen 4. Warten bis GUI erscheint Hinweis: Öffnen der GUI dauert beim Starten zum ersten Mal (1-2min)
Simulationsbetrieb	1. Vorkonfiguration im Skript main_PRG.py Instanzierung ServerController `dbcloudurl=srv_url, dbinstance = "Messung27_02", MLModelname="Random_Forest_model.pkl", simspeed = 0.5)` Hinweis: Konfigurieren in Zeile 179 (Name der DB Instanz, Modellname und Geschwindigkeit in s angeben) 2. Klicken auf Button „Simulation“ (Dauert ca. 5s) 3. Objekte aus der Datenbank werden iterativ verarbeitet und klassifiziert 4. Klicken auf Button „Stoppen“ unterbricht den Vorgang Hinweis: Falls das ML-Modell nicht geladen wird, den ganzen Skriptfolder öffnen (via VSC) Hinweis: Falls Simulation nicht startet, neu öffnen und als erstes auf Simulation Klicken

Daten empfangen	1. Netzwerk innerhalb der Skripte main.py und main_PRG.py konfigurieren (xxx ersetzen)
	main.py
	`if __name__ == "__main__":`
	`  print("Frei verfügbarer Speicher:", gc.mem_free())`
	`  client = TCPClient(host = "xxxx", port = xxx, ssid='xxx', pw='xxx', samplemode = 1)`
	Main_PRG.py
	`if __name__ == "__main__":`
	`  #Fenster Benutzeroberfläche`
	`  app = QtWidgets.QApplication(sys.argv)`
	`  #Cloud`
	`  username = urllib.parse.quote_plus('mongo')`
	`  password = urllib.parse.quote_plus('mongo')`
	`  srv_url = f'mongodb+srv://{username}:{password}@cluster21045.2xlz5.mongodb.net/?retryWrites=true&w=majority&appName=Cluster21045'`
	`  serverController = ServerController(host = "***", port = xxx, cloud=2, dbcloudurl=srv_url, dbinstance = "Messung22_02", MLModelname="Random_Forest_model.pkl")`
	2. (Optional) Angabe Abtastintervall oder Label (zur Klassifikation)
	3. Klicken auf „Daten empfangen“
	4. D1 Mini Resetten
	5. Auf Stoppen klicken, um Verbindung zu Unterbrechen

Anhang D: ML-Modelle Validierung

Trainingsdaten	Bestes Modell	Ergebnis
Alle Analysen mit Tiefpassfilter bei 40 HZ.		
Messung01_03	Random_Forest_model.pkl	Genauigkeit 92,5% Präzision 92,9% Status Simulation - Resultat Messung23_02 Stillstand: 28 von 68 richtig Schleudern: 70 von 84 richtig Waschen: 110 von 138 richtig Spülen: 74 von 136 richtig Insgesamt: 282 von 426 richtig! Status Simulation - Resultat Messung27_02 Stillstand: 9 von 17 richtig Schleudern: 51 von 51 richtig Waschen: 54 von 66 richtig Spülen: 38 von 42 richtig Insgesamt: 152 von 176 richtig! Status Simulation - Resultat Messung03_03 Stillstand: 0 von 51 richtig Schleudern: 65 von 73 richtig Waschen: 67 von 67 richtig Spülen: 75 von 75 richtig Insgesamt: 207 von 266 richtig!
Messung27_02 **Messung2**	LightGBM_model.pkl	Genauigkeit 90,7% Präzision 91,3% Status Simulation - Resultat Messung23_02 Stillstand: 16 von 68 richtig Schleudern: 71 von 84 richtig Waschen: 96 von 138 richtig Spülen: 98 von 136 richtig Insgesamt: 281 von 426 richtig! Status Simulation - Resultat Messung01_03 Stillstand: 25 von 61 richtig Schleudern: 64 von 67 richtig Waschen: 56 von 63 richtig Spülen: 60 von 74 richtig Insgesamt: 205 von 318 richtig! Status Simulation - Resultat Messung03_03 Stillstand: 49 von 51 richtig Schleudern: 65 von 73 richtig Waschen: 41 von 67 richtig Spülen: 65 von 75 richtig Insgesamt: 220 von 266 richtig!
Messung27_02 **Messung03_03** **Messung2**	LightGBM_model2.pkl	Genauigkeit 89,7% Präzision 89,8% Status Simulation - Resultat Messung01_03 Stillstand: 20 von 61 richtig Schleudern: 67 von 67 richtig Waschen: 44 von 63 richtig Spülen: 52 von 74 richtig Insgesamt: 183 von 318 richtig! Status Simulation - Resultat Messung23_02 Stillstand: 18 von 68 richtig Schleudern: 83 von 84 richtig Waschen: 100 von 138 richtig Spülen: 32 von 136 richtig Insgesamt: 233 von 426 richtig!

Anhang E: Datenbankinstanzen