

2022

EVALUIERUNG VON TECHNOLOGIEN ZUR SENSORBASIERTEN
NUTZUNGSERFASSUNG EINES AGILEN PROJEKTRAUMES

EVALUATION OF TECHNOLOGIES FOR SENSOR BASED USAGE
ANALYTICS OF AN AGILE WORKSPACE

Inhalt

1	Einleitung	3
1.1	Ziele und Motivation	3
1.2	Projektumfeld	4
1.3	Verwendete Technologien.....	4
1.3.1	Linux Applikationen	5
1.3.2	Python – Module	5
1.3.3	Arduino & C++ - Bibliotheken	6
1.3.4	Bluetooth Low Energy	7
1.3.5	MQTT	10
1.4	Struktur und Methodik	11
1.4.1	Sensorik.....	11
1.4.2	Rollen und Working Modes	14
1.4.3	Allgemeine Python Module.....	16
1.4.4	Edge Devices.....	18
1.4.5	Server	21
2	Umsetzung.....	25
2.1	Problem und Lösungsansatz.....	25
2.2	Evaluierung von Hardware und Sensorik.....	26
2.2.1	Microcontroller – ESP32 NodeMCU	26
2.2.2	Sensorik – Raum	27
2.2.3	Sensorik – se:lab High Desk	29
2.2.4	Sensorik – se:lab Board.....	31
2.2.5	Sensorik – se:lab Hopper.....	33
2.2.6	Edge-Devices – Raspberry Pi 3B+.....	35
2.3	Kommunikation zwischen ESP32-Boards und Edge Devices	35
2.3.1	Messen und Senden von Sensordaten	36
2.3.2	Empfangen und Weiterleiten von Sensordaten	38
2.4	People-Counting.....	41
2.4.1	Erstellen von Bildaufnahmen	41
2.4.2	Verarbeiten von Bildaufnahmen mit OpenPose	44
2.5	Speichern und Visualisieren der gewonnenen Sensordaten	48
2.6	Installation.....	52
3	Verschlüsselung und Absicherung der Kommunikation	53
3.1	SSL-Verschlüsselung	53
3.2	Absicherung MQTT	54
3.3	BLE Kennwörter	54

4	Ergebnisse	55
5	Aussichten.....	55
	Anhänge	Fehler! Textmarke nicht definiert.
6	Literaturverzeichnis.....	56
	Abbildungsverzeichnis	61
	Eigenständigkeitserklärung	Fehler! Textmarke nicht definiert.

1 Einleitung

Im Rahmen dieser Bachelorarbeit wird das Thema der Evaluierung von Technologien zur sensorbasierten Nutzungserfassung eines agilen Projektraumes bei der Firma Sedus Systems GmbH in Geseke als Konzept erarbeitet und als *Proof-of-Concept* (PoC) umgesetzt. Hierzu werden zunächst die Funktionseigenschaften von agilen Projektmöbeln der Firma Sedus aus der se:lab Modelreihe (*Abbildung 1*) untersucht, dessen mögliche sensorbasierte Nutzungsmessung evaluiert und anschließend implementiert. Diese Nutzungsmessungsdaten sollen an eine zentrale Datenbank gesendet werden können, welche als Grundlage für zukünftige Visualisierungen und Statistiken dienen soll.



Abbildung 1- Sedus Möbel der se:lab Modellreihe (Quelle: Sedus Stoll AG, 2020)

1.1 Ziele und Motivation

Die Corona-Pandemie hat in vielen Bereichen unsere Arbeitswelt teils komplett verändert. Für viele Unternehmen ist mobiles Arbeiten sowohl auf Arbeitnehmer- als auch auf Arbeitgeberseite keine große Hürde mehr, sodass Unternehmen nun vor der Herausforderung stehen ihre bisherigen Büroflächen an die neuen Arbeitsanforderungen und Gegebenheiten anzupassen. Das Büro wird immer mehr ein Ort, welcher von Mitarbeitern gemeinsam für agile Projektaktivitäten wie z.B. Workshops genutzt wird, während die Schreibtischarbeit aus dem Home-Office heraus erledigt werden kann. Für Facilitymanager eines Unternehmens, welche solche neuen agilen Projektflächen schaffen sollen, bleibt jedoch immer die Frage offen, ob diese neuen Flächen, mit den dazugehörigen Möbeln auch aktiv genutzt werden, oft unbeantwortet. Daher können meist weitere Optimierungen nur auf Schätzungen oder der Wahrnehmungen einiger Mitarbeiter als Entscheidungsgrundlage herangezogen werden. Daher soll untersucht werden, ob die Nutzung von agilen Projektmöbeln, sowie die Raumnutzung innerhalb eines agilen Projektraumes sensorgestützt gemessen und IT gestützt ausgewertet werden kann, sodass diese gewonnenen Daten über einen gewünschten Zeitraum betrachtet und visualisiert, als Entscheidungsgrundlage für Facilitymanager dienen können.

1.2 Projektumfeld

Sedus entwickelt, produziert und vertreibt ganzheitliche Büroeinrichtungen „Made in Germany“ seit mehr als 150 Jahren und vermarktet diese über 11 europäische Gesellschaften und eine außereuropäische Gesellschaft rund um den Globus.

Dank des hohen Engagements seiner ca. 1000 Mitarbeiter erwirtschaftete die Sedus Stoll Gruppe im Jahr 2019 einen Umsatz von über 210 Mio. Euro. [1]

Das Projekt, im weiteren Verlauf als erarbeitete Lösung bezeichnet, wird in einem bestehenden Projektraum für agiles Arbeiten am Produktionsort in Geseke durchgeführt, welcher sich auf Systemmöbel wie z.B. Tische, Schränke und Rollcontainer, sowie auch Raum-in-Raum-Systeme spezialisiert hat.

1.3 Verwendete Technologien

Die erarbeitete Lösung ist bis auf Ausnahme der Sensorprogrammierung in der Skriptsprache Python [2] programmiert und benötigt diese mindestens in der Version 3.8. Es wurde sich bewusst für Python entschieden, da Python auf Grund seiner simplen Syntax und seines Modulreichtums alle nötigen Bausteine bietet, um auf einfachem Wege auch komplexe Applikationen zu schreiben. Zudem können die meisten Applikationen sowohl auf Windows- als auch auf Linux-Systemen entwickelt werden und sind meistens auch auf diesen Systemen lauffähig. Die verwendeten Python-Module werden in den folgenden Abschnitten näher beschrieben und erläutert.

Als Entwicklungsumgebung für Python kommt *Visual Studio Code* [3] in der Version 1.62.0 von Microsoft zum Einsatz.

Alle Applikationen im Serverbereich werden auf Ubuntu Servern [4] in der LTS-Version 20.04, welche als Virtuelle Maschinen (VMs) auf einem VMware Cluster [5] in der Version 5.5 laufen, betrieben. Für die lokale Entwicklung wurde ebenfalls VMware Workstation Pro [6] in der Version 16.1.2 verwendet.

Im Edge Device Bereich kommen Raspberry Pi 3 B+ [7] Geräte mit dem aktuellem Raspberry Pi OS [8] in der Minimal Version (stand Januar 2022) zum Einsatz.

Alle verwendetet Sensoren werden über ESP32-Developmentboards der Firma AZ-Delivery [9] und mit Hilfe der Programmiersprache C++ innerhalb der Entwicklungsumgebung Arduino IDE in der Version 1.8.19 [10] programmiert und angesteuert.

Jegliche Kommunikation des Edge Device und Serverbereichs sind untereinander mittels *Transport Layer Security* (TLS [11]) in der Version 1.2 verschlüsselt. Lediglich die Kommunikation der Sensoren über *Bluetooth Low Energy* (BLE [12]) erfolgt unverschlüsselt.

1.3.1 Linux Applikationen

Zentrale Linux Applikationen der erarbeiteten Lösung bilden der MQTT-Broker *Mosquitto* [13] in der Version 2.0.14 und die Zeitreihendatenbank [14] *InfluxDB* [15] in der Version 2.1.1. Mosquitto dient als MQTT-Broker, welcher als zentrale Sammelstelle für das Empfangen und Bereitstellen von Sensordaten dient. Diese Sensordaten werden anschließend mit einem Zeitstempel versehen und in die Zeitreihendatenbank InfluxDB für ein Jahr gespeichert. Diese Daten können anschließend für spätere Analysen weiterverwendet werden.

Weitere zentrale Komponenten der erarbeiteten Lösung sind die Bibliotheken *OpenPose* [16] in der Version 1.7.0, sowie *OpenCV* [17] in der Version 4.5.4. Dabei dient OpenCV zum Erfassen und Verarbeiten von Bildaufnahmen in Python. OpenPose dient zum Erkennen und Zählen von menschlichen Strukturen auf einer Bildaufnahme. OpenPose kann hierbei lokal auf einem Linux System kompiliert und installiert oder optional innerhalb eines Docker Containers genutzt werden.

Für die Kommunikation mit den Sensoren per Bluetooth Low Energy dient das Linux Tool gatttool [18], welches als Teil des offiziellen Linux Bluetooth Stack *BlueZ* [19] vorhanden ist. Für die erarbeitete Lösung wurde *BlueZ* in der Version 5.63 verwendet.

1.3.2 Python – Module

Zentrale Komponenten der erarbeiteten Lösung im Server und Edge Device Bereich sind die Python-Module *paho-mqtt.client* [20] in der Version 1.6.1, sowie *influxdb_client* [21] in der Version 1.23.0. Das Modul *paho-mqtt.client* übernimmt hierbei eine zentrale Rolle für das Senden und Verarbeiten von Sensordaten per MQTT. Für den Import der gewonnenen Daten in eine Zeitreihendatenbank ist das Modul *influxdb_client* verantwortlich. Es sorgt dafür, dass alle gewonnenen Sensordaten passend für eine weitere Verarbeitung vorbereitet und übertragen werden.

Zudem bietet die erarbeitete Lösung im Server und Edge Device Bereich die Möglichkeit alle benötigten Programme und Dienste mittels Konfigurationsdateien im *YAML*- [22] oder *JSON*- [23] Format seiner Systemumgebung entsprechend anzupassen. Hierfür kommen die Module *json* in der Version 2.0.9 und *PyYAML* [24] in der Version 6.0 zum Einsatz. Je nach gewünschtem Konfigurationsdateiformat liefern diese Module ein Python Dictionary [25] mit allen gewünschten Konfigurationen, welche von den benötigten Programmen weiterverarbeitet wird, zurück.

Da die erarbeitete Lösung ebenfalls ein Personenzählssystem (engl. People Counter [26]) auf Basis von Bildaufnahmen integriert hat, werden hierfür Module für das Handling solcher Daten benötigt. Zum Bewältigen dieser Aufgabe kommen die Python-Module *opencv-python* [27] in der Version 4.5.3.56, *Pillow* [28] in der Version 8.4.0, sowie der Python-Wrapper *pyopenpose* für die OpenPose-Bibliothek in der Version 1.7.0 zum Einsatz. *Opencv-python* dient unter Anderem zum einfachen Aufnehmen und Ansteuern von USB-Kameras, sowie dem Verarbeiten dieser Aufnahmen innerhalb von Python. Für die Konvertierung verschiedener Bildformate wird *Pillow* verwendet. *Pyopenpose* dient zum Nutzen von OpenPose aus Python heraus und ermöglicht das Erkennen und Zählen menschlicher Strukturen auf Bildaufnahmen.

Das Personenzählssystem ist mittels einer simplen, selbstentwickelten *REST-API* auf Basis des Python-Moduls *Flask* [29] in der Version 2.0.2 als externer API-Endpunkt nutzbar. Zum Ansprechen dieser Schnittstelle wird *requests* [30] in der Version 2.26.0 verwendet.

Zur Kommunikation mit den Sensoren per Bluetooth Low Energy werden die Python-Module *bleak* in der Version 0.12.1 und *pexpect* [31] in der Version 4.8 benötigt. *Bleak* dient hierbei zum Empfangen

von Sensordaten, welche diese in festgelegten Intervallen scannt. Da die erarbeitete Lösung ebenfalls das Abändern dieser Intervalle vorsieht, kommt *pexpect* zum Einsatz. *Pexpect* steuert hierbei die Sensoren bei Bedarf mit dem Programm *gatttool* aus der offiziellen Linux Bluetooth Stack *BlueZ* an.

1.3.3 Arduino & C++ - Bibliotheken

Im Gegensatz zu den anderen Bestandteilen der erarbeiteten Lösung ist die Ansteuerung der Sensoren in der Programmiersprache C++ innerhalb der *Arduino IDE* in der Version 1.8.15 realisiert. Die Sensoren werden von einem ESP32-Entwicklungsboard angesteuert und ausgelesen. Es wurde sich hierbei bewusst für die Entwicklung in der *Arduino IDE* entschieden, da es für diese Entwicklungsumgebung bereits eine Vielzahl von Bibliotheken, Dokumentationen und Beispiele für das Ansteuern verschiedenster Sensoren gibt.

Die Arbeitsweise der sensorgestützten Datenerhebung ist so konzipiert, dass die ESP32-Entwicklungsboards in vordefinierten Zeitintervallen oder durch eine messbare Aktivität eines Sensors aus einem Ruhemodus aufwachen, die angeschlossene Sensorik ansprechen, auswerten und diese gemessenen Werte per Bluetooth Low Energy in Form eines Beacons für einige Sekunden per Broadcast senden. Danach gehen die ESP32-Entwicklungsboards wieder zurück in den Ruhemodus (*Deep Sleep*).

Hierbei spielen die BLE-Bibliotheken [32] und die Bibliothek *esp_sleep* [33] der Firma Espressif Systems eine zentrale Rolle.

Zum Ansteuern und Auslesen der Sensorik kommen unter anderem die Bibliotheken *DHT* [34], *Adafruit_CCS811* [35], *Adafruit_MP6050* [36] sowie *Adafruit_VL53L0X* [37] der Firma Adafruit Industries zum Einsatz.

Mittels *DHT* wird ein *DHT11* [38] Sensor, welcher sowohl die Umgebungstemperatur als auch die Luftfeuchtigkeit misst, angesprochen. *Adafruit_CCS811* wird für die Ansteuerung eines *CCS811* [39] Luftqualitätssensors genutzt. Die Bibliothek *Adafruit_MP6050* wird für die Ansteuerung eines *MPU6050* [40] Gyroskops verwendet, welcher die Ausrichtung des Sensors, sowie die Raumtemperatur messen kann. *Adafruit_VL53L0X* ermöglicht das Ansteuern eines *VL53L0X* Time-of-Flight-Entfernungssensor [41], welcher Laserbasiert die Entfernung zu einem Objekt messen kann.

1.3.4 Bluetooth Low Energy

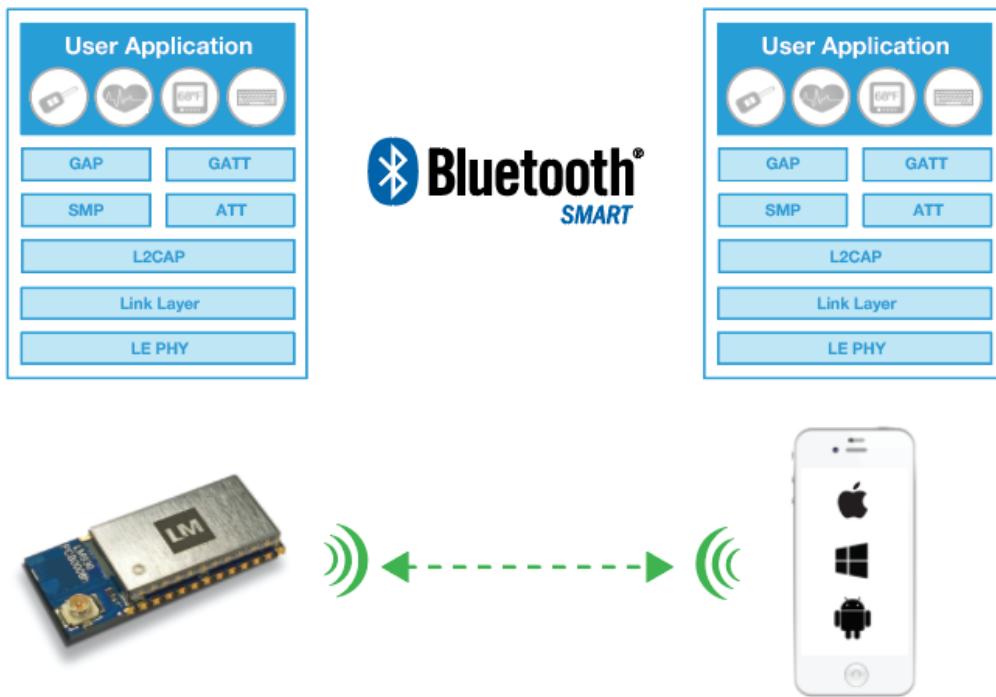


Abbildung 2 - Bluetooth Low Energy aka Bluetooth Smart (Quelle: wiki.lm-technologies.com)

Bluetooth Low Energy (BLE) ist eine sehr stromsparende Funktechnologie für Reichweiten bis zu 50 Metern, welche auf Bluetooth Classic basiert. *BLE* wurde speziell für Niedrigenergie-Lösungen im Bereich von Kontroll- und Monitoring Anwendungen entworfen. *BLE* ist seit der Bluetooth 4.0 Spezifikation ein fester Bestandteil des Industriestandards Bluetooth. Anwendung findet *BLE* in vielen verschiedenen Bereichen wie z.B. Haushaltselektronik, Medizin oder dem Sicherheitsbereich.

Das Low-Energy-Bluetooth-Protokoll hat mehrere Schichten und Strukturen, von denen die für die erarbeitete Lösung wichtigsten Elemente kurz in den folgenden Abschnitten erläutert werden[42].

1.3.4.1 GAP-Protokoll

GAP (Generic Access Profile) steuert die Sichtbarkeit, sowie den Verbindungsauflauf eines Bluetooth Gerätes zu anderen Teilnehmern.

In *GAP* können Geräte eine der folgenden vier Rollen annehmen:

Broadcaster: Ein *Broadcaster* überträgt Advertising Pakete (Broadcast) mit dessen Hilfe es von *Observern* gefunden werden kann. *Broadcaster* übertragen nur Advertising Pakete und können sich nicht mit anderen Geräten verbinden.

Observer: Ein *Observer* scannt die Umgebung nach *Broadcaster*-Geräten und meldet die Scan-Informationen an eine beliebige Applikation weiter. Es kann nur Scananfragen senden und keine Verbindung zu anderen Geräten herstellen.

Peripheral Device: Hierbei handelt es sich in der Regel um kleine Geräte mit geringem Stromverbrauch, die sich mit anderen Geräten verbinden können. Nachdem eine Verbindung hergestellt wurde, werden diese Geräte als Slave betrachtet (z. B. Sensoren).

Central Device: *Central Devices* starten und verwalten eine Verbindung zu anderen Geräten und arbeiten nach dem Verbindungsauftbau selbstständig. Bei diesen Geräten handelt es sich normalerweise um Geräte mit einer hohen Speicher- und Rechenkapazität, wie z.B. Mobiltelefone oder Tablets (*Abbildung 2*).

Unter den genannten Rollen sind für die erarbeitete Lösung peripherie und zentrale Rollen die Wichtigsten. Wenn sich Peripheriegeräte mit zentralen Geräten verbinden wollen, legen sie zunächst ein Advertising Intervall für die herzustellende Verbindung fest. Während dieser Intervalle beginnen sie mit dem Senden von Advertising Paketen (einmal pro Intervall). Nachdem das zentrale Gerät Pakete empfangen hat sendet es eine Antwort an das Peripheriegerät und eine Verbindung wird aufgebaut (*Abbildung 3*).

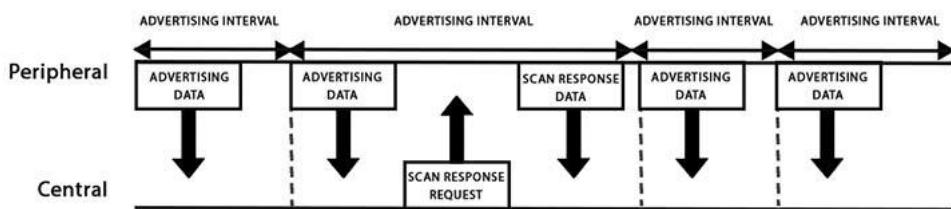


Abbildung 3 - BLE Advertising Data Diagramm (Quelle: electropeak.com)

Sobald eine Verbindung hergestellt wurde sendet das Peripheriegerät keine Pakete mehr und andere zentrale Geräte können das Peripheriegerät nicht mehr erkennen. Nachdem die Verbindung per *GAP-Protokoll* hergestellt wurde, ist das Senden und Empfangen von Daten nur noch über das *GATT-Protokoll* möglich.

1.3.4.2 GATT-Protokoll

Wie bereits beschrieben können sich Geräte per *GAP* gegenseitig erkennen und eine Verbindung aufbauen. Die anschließende Kommunikation, sprich das Senden und Empfangen von Daten, wird mit *GATT* (*Generic Attribute Profile*) realisiert. *GATT* definiert Methoden zum Senden und Empfangen von Daten zwischen zwei verbundenen Geräten über *BLE*. Die Datenübertragung im *GATT* erfolgt durch die beiden Konzepte *Services* und *Characteristics*.

GATT verwendet *ATT* (*Attribute Protocol*) um Daten zu senden und zu empfangen. Es speichert die Dienste und Merkmale und die dazugehörigen Daten in einer Tabelle (Lookup Table - LUT).

Der *GATT-Verbindungsmechanismus* ähnelt der Server- und Client-Architektur in Netzwerken. In dieser Struktur kann jedes Peripheriegerät ein Server sein und zentrale Geräte können als Clients betrachtet werden. Jedes Peripheriegerät (Server) hat eine eigene *ATT-Tabelle* mit gerätspezifischen *Characteristics* und *Services*.

1.3.4.3 Profile, Services und Characteristics

GATT umfasst ein Netzwerk, das aus verschiedenen Abschnitten besteht, die als *Profile*, *Services* und *Characteristics* bezeichnet werden. Diese sind hierarchisch miteinander verbunden.

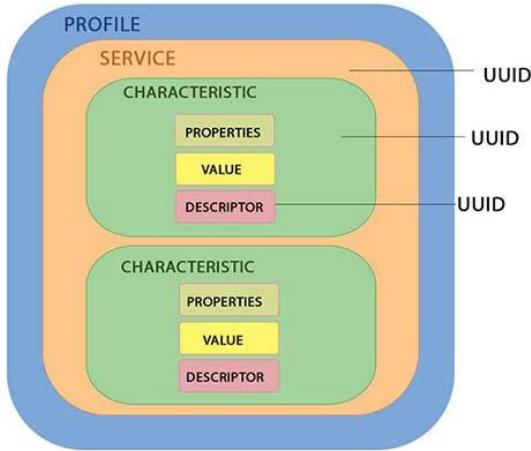


Abbildung 4 - Aufbau GATT (Quelle: electropoak.com)

Profile sind eine Reihe von *Services*, die von der *Bluetooth SIG (Bluetooth Special Interest Group)* oder Entwicklern von Peripheriegeräten definiert werden.

Services sind eine Reihe einfacher Informationen, wie z.B. Sensorwerte und werden in Unterkategorien unterteilt, die als *Characteristics* bezeichnet werden.

Verschiedene *Services*, wie z.B. das Lesen des Batteriestands, des Blutdrucks, der Herzfrequenz usw. werden von *Bluetooth SIG* definiert. Jeder Service hat dabei einen eindeutigen ID-Code, welcher als *UUID (Universally Unique Identifier)* bezeichnet wird. Eine *UUID* kann 16 Bit (für offizielle Services) oder 128 Bit (für Dienste anderer Hersteller) umfassen. Beispielsweise lautet die *16-Bit-UUID* für den Batteriestand *0x180F* [43].

1.3.4.4 Characteristics

Services können in ein oder mehrere *Characteristics* unterteilt werden. Zum Beispiel verfügen Beschleunigungssensoren über drei messbare Achsen-Koordinaten X, Y und Z, welche alle als eine einzelne *Characteristic* innerhalb eines *Service* dargestellt werden können. Ähnlich wie bei *Services* verfügen auch *Characteristics* über eine eindeutige *16- bzw. 128-Bit-UUID*.

1.3.5 MQTT

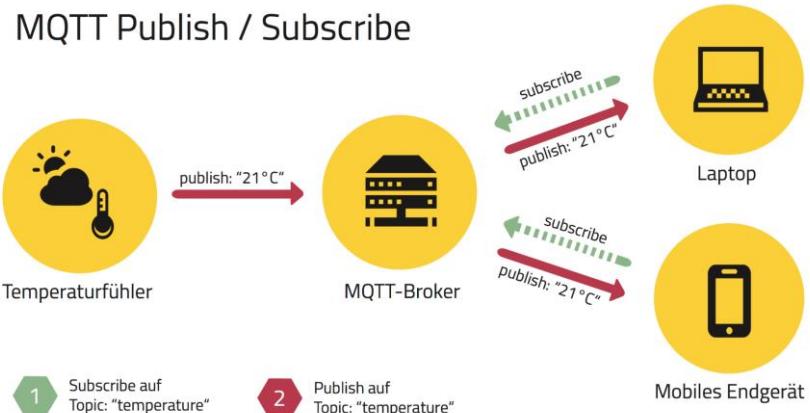


Abbildung 5 - Schaubild MQTT (Quelle: informatik-aktuell.de)

Message Queue Telemetry Transport oder kurz *MQTT* ist ein leichtgewichtiges Protokoll auf Basis von *TCP*, welches sich in den letzten Jahren zum de facto Standard für das *Internet of Things (IoT)* etabliert hat. Es wurde zur *M2M – (Machine to Machine)* – Kommunikation von IBM und Arcom Control Systems im Jahre 1999 zur Überwachung von Ölpielines entwickelt. Es ist darauf ausgelegt, dass es einfach zu implementieren ist, damit auch Geräte mit geringen Ressourcen oder einer instabilen Netzwerkverbindung vernetzt und für eine Kommunikation genutzt werden können [44].

Seit dem Jahr 2010 ist das Protokoll in der Version 3.1 unter einer freien Lizenz (*Royalty Free License*) veröffentlicht. Seit 2014 existiert *MQTT* in der Version 3.1.1 und seit 2018 [45] in der Version 5.0. Für die erarbeitete Lösung kommt *MQTT* in der Version 3.1.1 zum Einsatz, da es gegenüber der Version 5.0 keine Nachteile bietet und in vielen Bereichen immer noch die Standardversion ist.

MQTT arbeitet mit einem Client-Server-Prinzip. Einer der zentralen Aspekte von *MQTT* ist die ereignisgesteuerte Publish/Subscribe-Architektur. Teilnehmer welche Informationen an den *MQTT-Broker* senden, werden *Publisher* genannt. Teilnehmer welche Information des *MQTT-Brokers* Abrufen sind *Subscriber*. Bei *MQTT* gibt es keine Ende-zu-Ende-Verbindung, wie beispielsweise bei *HTTP* welche mit einer Request/Response-Architektur arbeitet. Stattdessen gibt es einen zentralen *Server (MQTT-Broker)*, zu welchem sich Sender und Empfänger von Daten gleichermaßen verbinden können. Nachrichten werden über sogenannte *Topics* gesendet (*published*) oder empfangen (*subscribed*). Ein *Topic* ist ein String, der eine URL-ähnliche Struktur aufweist und eine Art Betreff der Nachricht darstellt.

`testroom/sensor/room/temperature`

Abbildung 6 - MQTT-Topic

Abbildung 6 zeigt hierbei beispielsweise das *Topic* eines Temperatursensors im Projektraum.

Entscheidend hierbei ist, dass das *Topic* nicht die Adresse des Temperatursensors darstellt, sondern einen Kommunikationskanal, den der Temperatursensor nutzt um Nachrichten an den *MQTT-Broker* zu senden. Der *MQTT-Broker* überprüft anschließend, welche Interessenten (*Subscriber*) einen Kanal zum Empfangen dieser Daten geöffnet haben und leitet die Nachrichten an Diese weiter.

1.4 Struktur und Methodik

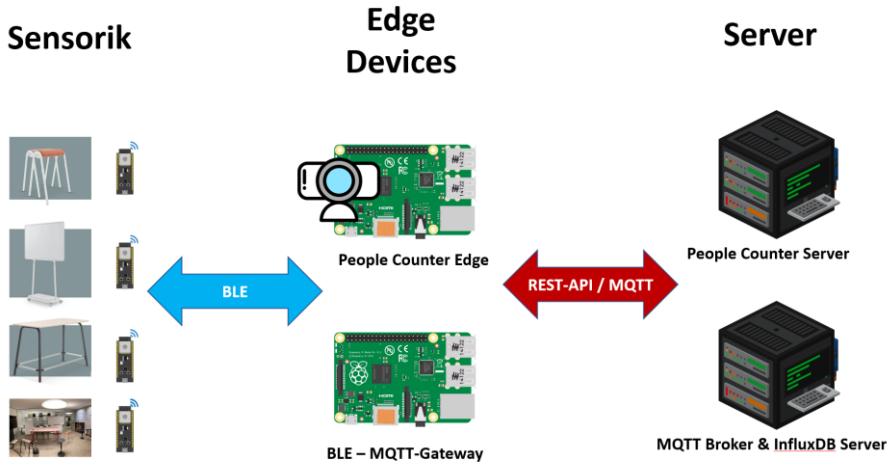


Abbildung 7 - Datenfluss und Kommunikation

Wie bereits in *Kapitel 1.3* beschrieben, unterteilt sich die erarbeitete Lösung in die Bereiche Sensorik, Edge Devices und Server. Abbildung 7 zeigt hierbei die jeweiligen Kommunikationswege und Übertragungstechniken/-protokolle, die zum Datenaustausch zwischen diesen Bereichen verwendet und im weiteren Verlauf noch näher beschrieben werden.

Die folgenden Abschnitte sollen dem Verständnis zum Aufbau und der Funktionsweise der einzelnen Bereiche dienen.

1.4.1 Sensorik

Arbeitsweise der Sensoren

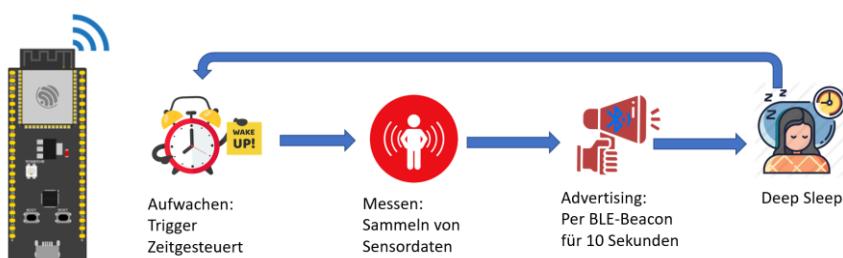


Abbildung 8 - Arbeitsweise Sensorik

Wie in Abbildung 8 skizziert, ist die Arbeitsweise der Sensoren in die vier Phasen Aufwachen, Messen, Advertising und Deep Sleep aufgeteilt, welche in diesem Abschnitt erklärt werden.

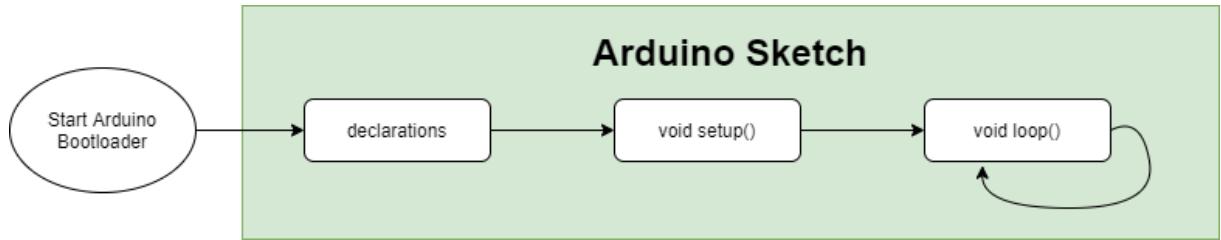


Abbildung 9 - Grundaufbau Arduino Sketch

Wie bereits in *Kapitel 1.3* erwähnt, ist die Programmierung der Sensorik mit der Arduino IDE realisiert. Innerhalb der Arduino IDE werden die Programme als Sketches (dt. Skizzen) bezeichnet. Abbildung 9 zeigt hierbei vereinfacht dargestellt den Grundaufbau eines jeden Arduino Sketches.

Nach dem Einschalten eines arduinofähigen Gerätes wird im ersten Schritt der Arduino Bootloader geladen, welcher das aufgespielte Arduino Sketch lädt und ausführt [46]. Ein Arduino Sketch kann in drei Phasen unterteilt werden. In der ersten Phase *declarations* werden Bibliotheken eingebunden und globale Variablen sowie Methoden definiert. Anschließend folgt die Methode *void setup()*. Diese Methode dient für Aufgaben, welche nur einmalig durchgeführt werden sollen. Danach folgt die Methode *void loop()* welche, wie der Name bereits vermuten lässt, sich daraufhin ständig wiederholt.

Auf Abbildung 9 bezogen ergibt sich hierbei, dass mit der Phase *Aufwachen* der Arduino Bootloader auf dem ESP32-Entwicklerboard gestartet und der jeweilige Sensor-Sketch geladen wird.

Hierbei kann das Aufwachen entweder zeitgesteuert oder durch einen Trigger erfolgen. Zeitgesteuert bedeutet hierbei, dass das ESP32-Entwicklerboard aus einem Deep Sleep Modus nach einer vorgegebenen Zeit „aufwacht“ und sein Programm/Sketch ausführt. Alternativ kann das Aufwachen durch einen Trigger erfolgen. Dies bedeutet, dass durch einen äußeren Einfluss, wie z.B. der Ausschlag eines Sensors, das ESP32-Entwicklerboard vorzeitig aus seinem *Schlaf-Modus* aufgeweckt wird.

Anschließend erfolgt, wie in Abbildung 10 auszugweise zu sehen, das Einbinden von benötigten Bibliotheken, sowie globaler Variablen und Methoden.

```

CHAIR_SENSOR
#include "sys/time.h"
#include "BLEDevice.h"
#include "BLEServer.h"
#include "BLEUtils.h"
#include "esp_sleep.h"

#define SERVICE_UUID          "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID   "beb5483e-36e1-4688-b7f5-ea07361b26a8"
#define SEAT_PIN 13

int GPIO_DEEP_SLEEP_DURATION = 10; // sleep 10 seconds and then wake up
RTC_DATA_ATTR static int working_mode;
RTC_DATA_ATTR static bool initialized;
int loop_counter = 0;

bool deviceConnected = false;
bool oldDeviceConnected = false;

```

Abbildung 10 - Auszug declaration "Chair_Sensor"

Die in Abbildung 8 dargestellten Phasen Messen, Advertising und Deep Sleep erfolgen anschließend in der *void setup()* Methode des Arduino Sketches. Grund dafür, dass diese Schritte nicht in der *void loop()* stattfinden sind, dass nur einmalig Sensordaten für einen gewissen Zeitraum gesammelt und

anschließend per Bluetooth Low Energy für 10 Sekunden übertragen werden sollen. Anschließend soll der Sensor sich wieder „schlafen legen“, bis er seine Arbeit auf ein Neues aufnimmt.

Nichts desto trotz ist die `void loop()`, wie in Abbildung 11 zu sehen, ebenfalls implementiert. Dies hat den Grund, dass es während der 10 Sekunden, in welcher die erfassten Sensordaten übertragen werden die Möglichkeit gibt, sich mit dem ESP32-Entwicklerboard per Bluetooth Low Energy zu verbinden, um die Dauer des Deep Sleep Modus variabel anzupassen. Hierbei dient die `void loop()` dazu, dass die maximale Dauer einer aufgebauten Bluetooth Low Energy Verbindung nicht mehr als 40 Sekunden (20*2000ms) beträgt. Damit soll verhindert werden, dass sich auf Grund einer nicht sauber geschlossenen Verbindung das ESP32-Entwicklerboard aufhängt und keine Daten mehr liefert.

```
void loop() {  
  
    delay(2000);  
    loop_counter++;  
  
    if(!deviceConnected || loop_counter>20){  
        Serial.printf("enter deep sleep loop\n");  
        esp_deep_sleep(1000000ULL * GPIO_DEEP_SLEEP_DURATION);  
    }  
  
}
```

Abbildung 11 - Void Loop

Nachdem vom jeweiligen ESP32-Entwicklerboard alle vorhandenen Sensoren ausgelesen worden sind, werden diese Daten in einem *Eddystone Bluetooth Low Energy Beacon* [47] in einem *TLM-Frame* zusammengefügt (Abbildung 12) und anschließend für 10 Sekunden per BLE Advertising publiziert. In dieser Zeit können diese Beacon von den Edge Devices aufgenommen und verarbeitet werden.

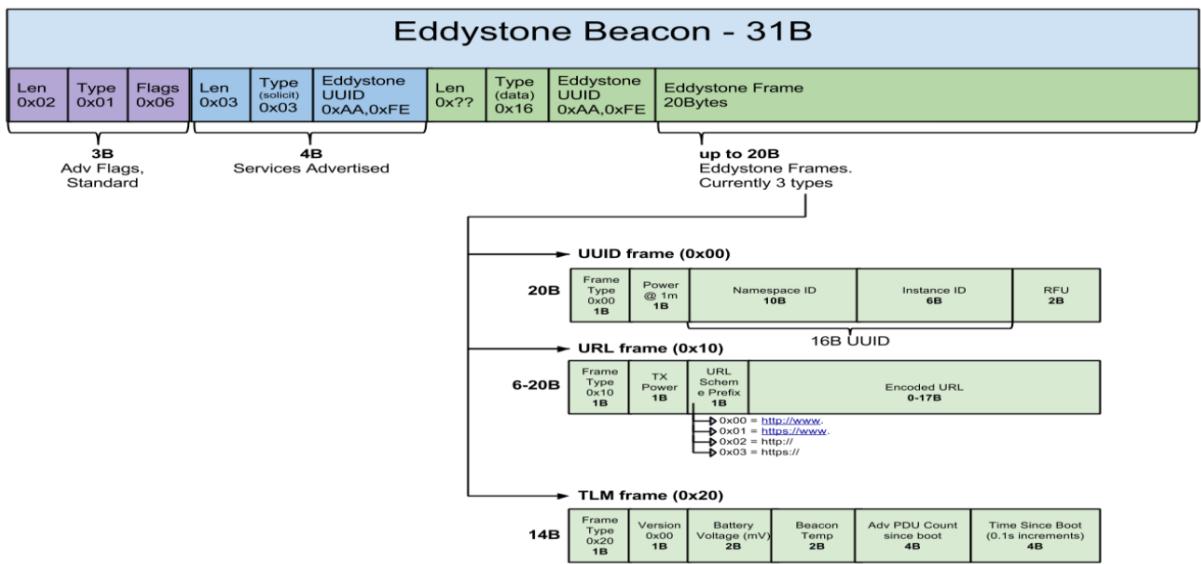


Abbildung 12 – Definition eines Eddystone BLE Beacon (<https://os.mbed.com/>, 2015)

Abbildung 12 skizziert hierbei beispielhaft den Aufbau und die Definition [48] eines *Eddystone BLE Beacons* in einem Datagramm.

1.4.2 Rollen und Working Modes

Um die Datenerhebung der Sensoren möglichst effizient zu gestalten, verfügt die erarbeitete Lösung über ein Rollen- und Working Mode -System, welches die Messintervalle steuert und regelt.

Die erarbeitete Lösung verfügt über folgende drei Working Modes:

- Mode 0 – Sleep (60 Minuten)
- Mode 1 – Regular (5 Minuten)
- Mode 2 – Burst (1 Minute)

Die Working Modes definieren, wie lange sich die ESP32-Boards in den Deep Sleep Modus versetzen sollen. Befindet sich z.B. keine Person im Projektraum, so ist es nicht nötig, dass jede Minute Daten ermittelt und übertragen werden. Dies gilt z.B. auch für die Zeit über Nacht, in der i.d.R keine Aktivitäten im Raum stattfinden. Der *Mode 0* regelt genau dieses Szenario, indem das ESP32-Board für 60 Minuten in den Deep Sleep Modus versetzt wird. Im Gegensatz zum *Mode 0* steht der *Mode 2*, welcher dafür sorgt, dass so viele Daten wie möglich erhoben und gesendet werden. Dies ist unter anderem für die Messung der Raumeigenschaften wie Luftqualität und Raumtemperatur interessant, um festzustellen wie sich diese Werte bei Workshops oder Besprechungen minutenweise verändern. Der *Mode 1* dient als optionaler Standard-Modus um Messungen über ein Intervall von 5 Minuten zu realisieren. Dieser Modus wird in der erarbeiteten Lösung nicht verwendet, bleibt jedoch für mögliche zukünftige Szenarien weiter implementiert.

```
208 //Get Working Mode and set duration for Deep Sleep
209 int get_deep_sleep_duration(int wm) {
210
211     if (wm == 1) {
212         Serial.println("Set Sleep for 300 Seconds (5 Minutes)");
213         return 300;
214     }
215     if (wm == 2) {
216         Serial.println("Set Sleep for 60 Seconds (1 Minute)");
217         return 60;
218     }
219     Serial.println("Set Sleep for 3600 Seconds (1 Hour)");
220     return 3600;
221 }
222 }
```

Abbildung 13 - Working Modes im Arduino Sketch

Die Zeitintervalle der einzelnen Working Modes sind dabei in den Sketches der ESP32-Boards hart kodiert (*Abbildung 13*).

Wenn die Dauer des Deep Sleeps abgelaufen ist, oder das ESP32-Board durch einen Trigger Sensor geweckt worden ist, nimmt das ESP32 Board seine Arbeit auf und sendet seine erhobenen Daten als Advertising Package (BLE-Broadcast). Während dieses Sendevorgangs ist es möglich den Working Mode des ESP32-Boards zu verändern. Der momentan gültige Working Mode ist serverseitig gespeichert und wird über die Edge-Devices an die ESP32-Boards per BLE übertragen und aufgespielt.

Um festzulegen welcher Working Mode momentan gültig ist, verfügt die erarbeitete Lösung über ein Rollensystem, welches diesen bei der messbaren Nutzung bestimmter Sensoren ändern kann.

Die erarbeitete Lösung verfügt dafür über folgende Rollen:

- Follower
- Influencer
- Switchmaster

Als *Follower* gelten alle ESP32-Boards, welche nur den aktiven Working Mode annehmen, diesen jedoch nicht ändern können.

Influencer können dagegen bei einer erfolgreichen Nutzungsmessung den Working-Mode erhöhen und so beispielsweise den Working Mode zentral von 0 auf 2 hochsetzen, sodass alle *Follower* diesen Modus annehmen.

Beim *Switchmaster* handelt es sich um eine Rolle, welche dazu berechtigt ist, den momentanen Working Mode wieder runterzusetzen. Dies geschieht z.B., wenn der People Counter nach drei Messungen immer noch keine Personen erkannt hat. Somit gilt der Raum per festgelegter Definition auch als nicht genutzt, weswegen auch nicht ständig Daten erhoben werden müssen.

```
#General Config of the Serverbehavior
general_config:
    start_hour: 6
    end_hour: 22
    influencer:
        - testroom/sensor/people_counter/usage
        - testroom/sensor/room/usage
    default_workmode: 0
    switch_master :
        - testroom/sensor/people_counter/usage
    switch_max_counter: 3
    switch_back_mode : 0
```

Abbildung 14 – Definition von Influencer und Switchmaster in mqtt-to-influx-broker Konfigurationsdatei

Die Definition der beschriebenen Rollen erfolgt in der Konfigurationsdatei des *mqtt-to-influx-broker*. Wie in Abbildung 14 zu sehen, werden die Rollen über *MQTT-Tops* definiert. Dabei ist es auch möglich, dass ein *MQTT-Topic* mehrere Rollen haben kann. Zudem wird in dieser Konfigurationsdatei festgelegt, nach wie vielen negativen Messungen ein *Switchmaster* die ESP32-Boards in welchen Working Mode versetzt. Des Weiteren können über die Parameter *start_hour* und *end_hour* Zeiträume definiert werden, in denen das Ändern der Working Modes möglich ist. Außerhalb dieser Zeiten wird der Working Mode für alle Sensoren auf Mode 0 gesetzt, damit diese z.B. Über Nacht so wenig arbeiten wie möglich.

1.4.3 Allgemeine Python Module

Wie bereits in Kapitel 1.3 erwähnt, ist die erarbeitete Lösung im Server und Edge-Device Bereich in der Skriptsprache Python geschrieben.

```
# mod_sendData.py > ...
1  from datetime import datetime
2  from influxdb_client import InfluxDBClient, Point, WritePrecision
3  from influxdb_client.client.write_api import SYNCHRONOUS
4  import paho.mqtt.client as mqtt
5  #Zum deaktivieren der SSL-CERT WARNINGS
6  import requests
7  from requests.packages.urllib3.exceptions import InsecureRequestWarning
8  requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
9
10 #Send Sensordata via Influx-Rest-API
11 def sendInfluxData(data,timestamp,
12                     mqtt_topic=None,
13                     influx_server="192.168.178.36",
14                     influx_port=8086,
15                     influx_bucket="sensor_data",
16                     influx_org="sedus",
17                     influx_token="XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
18                     use_tls = False
19 >             ): ...
56
57
58 #Send Sensordata via MQTT (Unsecure)
59 > def sendMQTTData(MQTT_Server,topic,payload,port=1883,timeout=60,mqtt_retain=False): ...
64
65
66 #Send Sensordata via MQTT (Secure)
67 > def sendMQTTData_secure(MQTT_Server,topic,payload,mqtt_ca_cert,mqtt_secure_user,mqtt_secure_pw,port=8888,timeout=60,mqtt_reta
76
77 #testing
78 if __name__ == '__main__':
79     sendInfluxData(15,1637959636,mqtt_topic='testroom/sensor/se_lab_hopper_badeaffebadeafffe/usage')
```

Abbildung 15 - mod_sendData.py

Wie aus Abbildung 7 zu entnehmen, können diese Bereiche sowohl über REST-API's als auch über MQTT miteinander kommunizieren. Abbildung 15 zeigt hierbei einen Auszug aus dem selbstentwickelten Modul *mod_SendData.py*, welches sowohl auf Server als auch auf Edge-Device Seite zum Einsatz kommt und die Übertragung von Sensordaten ermöglicht und vereinheitlicht.

Eine ebenfalls zentrale Rolle auf allen Server und Edge-Devices spielt das selbstentwickelte Modul *mod_read_config.py* (Abbildung 18).

Dieses Modul liest eine gewünschte Konfigurationsdatei im YAML- oder JSON-Format ein und liefert die gewünschten Konfigurationswerte als Python-Dictionary-Objekt (Abbildung 9 – Zeile 17) zurück. Dieses kann anschließend innerhalb eines Python-Programms verwendet werden.

```
8  import mod_read_config as read_config
9
10 #Runs on MQTT-Server as a MQTT to InfluxDB broker
11
12 #read Config-File
13 config_file = "server_config_pattern.yaml"
14 config = read_config.get_server_config(config_file)
15
16 #Global values
17 WORK_START_HOUR = config['general_config']['start_hour']
```

Abbildung 16 - Einlesen von Konfigurationsdateien

Abbildung 17 zeigt beispielhaft eine YAML-Konfigurationsdatei für die Serverkomponente des MQTT und InfluxDB Brokers.

```
! server_config_pattern.yaml
C: > Users > sdialor > Documents > Bachelorarbeit > Project_Files > Source > Server_and_Edge > MQTT-Server > ! server_config_pattern.yaml
1 #General Config of the Serverbehavior
2 general_config:
3   start_hour: 6
4   end_hour: 22
5   influencer:
6     - testroom/sensor/people_counter/usage
7     - testroom/sensor/room/usage
8   default_workmode: 0
9   switch_master :
10    - testroom/sensor/people_counter/usage
11   switch_max_counter: 3
12   switch_back_mode : 1
13
14 #Config for MQTT-Server
15 mqtt_config:
16   mqtt_server: 192.168.178.36
17   mqtt_port: 1883
18   mqtt_timeout: 60
19 > mqtt_subs: ...
20   mqtt_qos: 0
21   mqtt_tls_enable: false
22   mqtt_tls_port : 8888
23   mqtt_tls_ca_cert_file: my_ca.crt
24   #for future use
25   #mqtt_tls_ca_cert_file : /etc/mosquitto/certs/my_ca.cert
26   #mqtt_tls_server_cert_file : /etc/mosquitto/certs/my_server.cert
27   #mqtt_tls_server_key_file : /etc/mosquitto/certs/my_server.key
28   mqtt_tls_user_name: InfluxUser
29   mqtt_tls_user_pass: InfluxUserPass
30
31 #Config for InfluxDb
32 influxdb_config:
33   influxdb_server: 192.168.178.36
34   influxdb_port: 8886
35   influxdb_tls_enable: true
36   influxdb_api_token: InfluxUserToken
37   influx_bucket: sensor_data
38   influx_org: sedus
```

Abbildung 17 - Konfiguration in YAML

Abbildung 18 zeigt auszugweise die Verarbeitung einer YAML-Konfigurationsdatei.

```
mod_read_config.py > ...
1 import json
2 import yaml
3 import os.path
4
5 def get_server_config(config_file="server_config.yaml"):
6
7     #Check if Config file exists
8     if os.path.isfile(config_file):
9         #get the filetype (yaml or json)
10        config_type = config_file.split(".")[-1]
11
12        #handling for yaml-config-files
13        if config_type.lower() == "yaml":
14            try:
15                yaml_config = open(config_file,'r')
16                yaml_server_config = yaml.full_load(yaml_config)
17                #return config
18                return yaml_server_config
19            except:
20                print("ERROR: \t{} invalid Yaml-file".format())
21                return "error in yaml handler"
22            finally:
23                yaml_config.close()
24
25        #handling for json-config-files
26        elif config_type.lower() == "json": ...
27
28        #handling if config-file does not exists
29        else:
30            print("ERROR:\tFile {} not found or does not exist!".format(config_file))
31            return "error in file handler"
```

Abbildung 18 - mod_read_config.py

1.4.4 Edge Devices

Die Edge Devices haben innerhalb der erarbeiteten Lösung die Aufgabe, Sensordaten zu empfangen und passend formatiert an die vorgesehenen Serversysteme weiterzuleiten.

Wie bereits erwähnt handelt es sich hierbei um einen *People Counter* und ein *BLE – MQTT – Gateway*, welche in den folgenden Abschnitten näher beschrieben werden.

1.4.4.1 People Counter

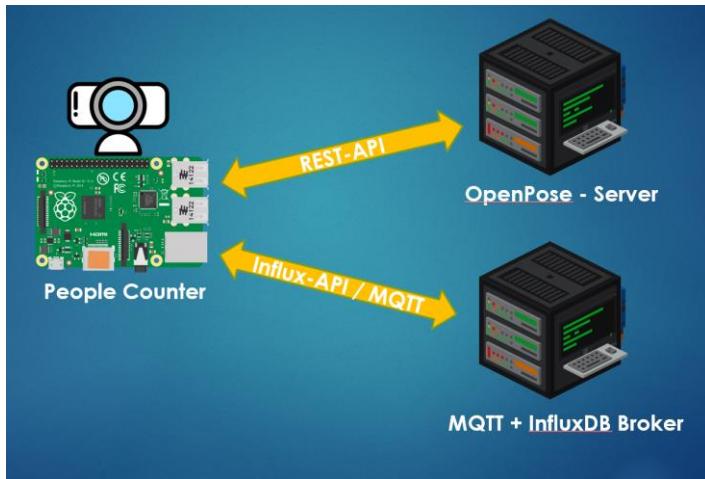


Abbildung 19 - Arbeitsweise People-Counter

Der People Counter ist innerhalb der erarbeiteten Lösung als optischer Sensor im Client-Server-Betrieb konzipiert. Dabei dient ein *Raspberry Pi 3B+* mit einer angeschlossenen USB-Webcam auf der Client-Seite als „Auge in den Raum“. Im Projektordner „*PEOPLE_COUNTER_EDGE*“ (Abbildung 20) befindet sich, neben den bereits erwähnten Modulen *mod_read_config.py* und *mod_sendData.py*, das Hauptprogramm *people-counter-client.py*, sowie die dazugehörige Konfigurationsdatei im *YAML*-Format. Zusätzlich befindet sich im Projektordner mit *install_people_counter.sh* ein BASH-Skript, welches mit Root-Rechten gestartet, alle nötigen Komponenten installiert und zudem den People-Counter als Dienst einrichtet.

```
▽ PEOPLE_COUNTER_EDGE
$ install_people_counter.sh
⠃ mod_read_config.py
⠃ mod_sendData.py
! people_counter_config.yaml
⠃ people-counter-client.py
≡ people-counter-client.service
```

Abbildung 20 - Projektordner "PEOPLE_COUNTER_EDGE"

Aufgabe der *people-counter-client.py* ist es, in zeitlich regelmäßigen Intervallen Aufnahmen vom Raum zu machen und diese zur Auswertung an einen *OpenPose* Server per *REST-API* zu senden. Zum Erstellen der Aufnahmen wird *OpenCV* verwendet, welches Bilddaten innerhalb eines NumPy-Arrays speichert.

```

#OPENPOSE_WORKING_MODES:
# ->> "FILE" : Macht ein Foto und überträgt das Bild an Server zum Auswerten
# ->> "JSON" : Macht ein Bild und codiert dieses als Base64 innerhalb einer JSON
OPENPOSE_WORKING_MODE = config['openpose_config']['working_mode']

if OPENPOSE_WORKING_MODE == "FILE":
    OPENPOSE_SERVER = "{}/FilePeopleCounting".format(OPENPOSE_SERVER)
elif OPENPOSE_WORKING_MODE == "JSON":
    OPENPOSE_SERVER = "{}/JsonPeopleCounting".format(OPENPOSE_SERVER)
else:
    print("NO WORKING MODE GIVEN!")
    exit(-1)

```

Abbildung 21 - OpenPose Working Modes

Der People Counter unterstützt, wie in Abbildung 21 zu sehen, zwei Arbeitsmodi (engl. Working Modes).

Im Working Mode *JSON* wird die Aufnahme zunächst in ein Base64 String umkodiert und anschließend als JSON-Dataframe an den *OpenPose Server* übertragen. Vorteil dieses Working Modes ist, dass für die Verarbeitung der Bildaufnahmen weder auf Edge-Device noch auf Server Seite eine Bilddatei gespeichert werden muss und somit auch nicht einsehbar ist. Somit ist dieser Working Mode DGSVO-Konform [49].

Im Working Mode *FILE* wird die Bildaufnahme zunächst als JPG-Datei gespeichert und anschließend als Binärdatei an den *OpenPose Server* übertragen und ausgewertet. Nach der Verarbeitung durch den *OpenPose Server* werden die Bilddateien sowohl auf Server- als auch auf Edge-Device-Seite wieder gelöscht.

Der *FILE Modus* stammt aus einer sehr frühen Entwicklungsphase der erarbeiteten Lösung und wird im Standard nicht weiter eingesetzt. Jedoch bietet dieser Modus die Möglichkeit auch anderen Low-End Systemen, wie z.B. einer ESP32-CAM, mit Hilfe von Bildaufnahmen die Dienste des *OpenPose Servers* in Anspruch zu nehmen, weswegen dieser Modus zunächst implementiert bleibt.

```

218     #Behavior of Send Modes
219
220     if SEND_MODE.upper() == "MQTT":
221         mqtt_count_payload = "{}:{}".format(timestamp,result)
222
223     if (int(result)>0): ...
224     else: ...
225
226     if MQTT_SECURE == False: ...
227     else:
228         mod_sendData.sendMQTTData_secure(MQTT_SERVER, ...
229             time.sleep(0.2)
230         mod_sendData.sendMQTTData_secure(MQTT_SERVER, ...
231             time.sleep(0.2)
232
233     elif SEND_MODE.upper() == "INFLUX": ...
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280

```

Abbildung 22 - Send Modes

Nachdem der *OpenPose-Server* eine Aufnahme verarbeitet hat, liefert dieser eine Anzahl der erkannten Personen an den *People Counter* zurück. Dieses Ergebnis kann nun wahlweise per *MQTT* an einen *MQTT-Broker* gesendet oder alternativ direkt über die *Influx-API* in eine *InfluxDB* geschrieben werden. Dieser *Send Mode* kann innerhalb der Konfigurationsdatei definiert werden.

1.4.4.2 BLE – MQTT – Gateway

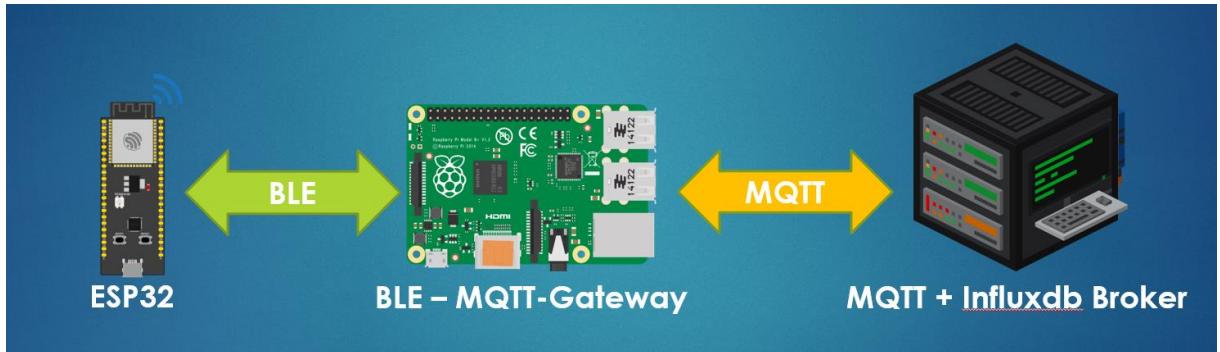


Abbildung 23 - Datenfluss BLE-MQTT-Gateway

Das *BLE-MQTT-Gateway* dient innerhalb der erarbeiteten Lösung als Bindeglied zwischen den ESP32-Entwicklerboards, welche die gemessenen Sensordaten per Bluetooth Low Energy übertragen und dem MQTT-Broker.

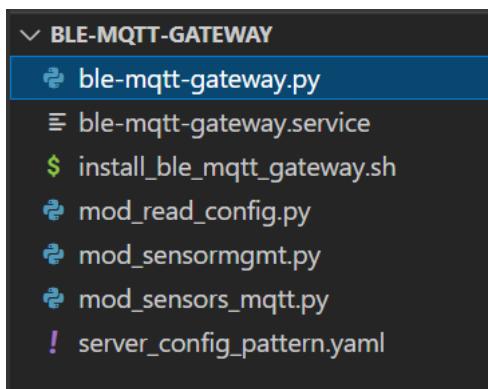


Abbildung 24- Projektordner "BLE-MQTT-GATEWAY"

Abbildung 24 zeigt den Projektordner „BLE-MQTT-GATEWAY“ , welcher alle nötigen Komponenten für die Installation und den Betrieb des Gateways bereitstellt.

Das Hauptprogramm ist die Datei *ble-mqtt-gateway.py*, welche die Umgebung nach Bluetooth-Paketen abscannt, die benötigten *Beacon-Daten* herausfiltert, aufbereitet und an den *MQTT-Broker* weiterleitet. Dabei kommen die selbstentwickelten Module *mod_sensors_mqtt.py* und *mod_sensormgmt.py* zum Einsatz.

mod_sensors_mqtt.py kümmert sich um die Aufbereitung und das Versenden der erhaltenen Sensordaten an den *MQTT-Broker*.

mod_sensormgmt.py dient dazu, bei Bedarf eine Kommunikation mit den ESP32-Entwicklerboards aufzubauen, um z.B. den Working Mode abzuändern.

Wie in den anderen Bestandteilen der erarbeiteten Lösung ist auch diese Komponente per Konfigurationsdatei im *YAML*-Format individuell konfigurierbar.

1.4.5 Server

Wie bereits beschrieben besitzt die erarbeitete Lösung zwei Serverkomponenten in Form eines *OpenPose-Servers* und eines *MQTT – InfluxDB Brokers*, dessen Funktionsweisen in den folgenden Abschnitten kurz erläutert werden.

1.4.5.1 OpenPose – Server

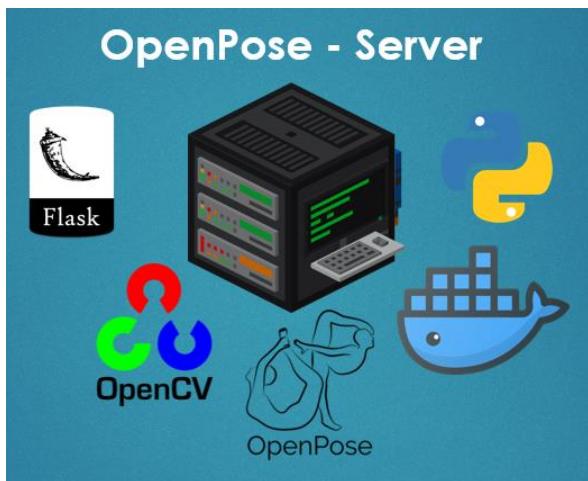


Abbildung 25 - Komponenten "OpenPose-Server"

Mittels des Python-Moduls *FLASK* ist auf dem *OpenPose-Server* eine einfache *REST-API* implementiert, welche es ermöglicht, menschliche Körper, wie in Abbildung 26 zu sehen, auf Bildaufnahmen zu ermitteln und zu zählen.

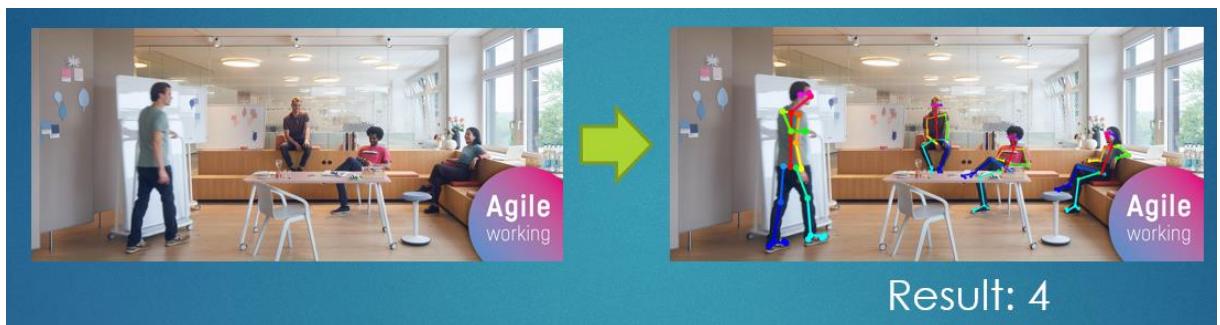


Abbildung 26 - OpenPose Erkennung

Zum Ermitteln dieser Daten kommt *OpenPose* zum Einsatz. Bei *OpenPose* handelt es sich um eine Bibliothek zum Erkennen mehrerer Personen in Echtzeit. *OpenPose* kann dabei innerhalb der erarbeiteten Lösung in einem *Docker-Container* oder native auf dem Server kompiliert und ausgeführt werden. Für das Handling der Bilddaten wird die Bildverarbeitungsbibliothek *OpenCV* verwendet.

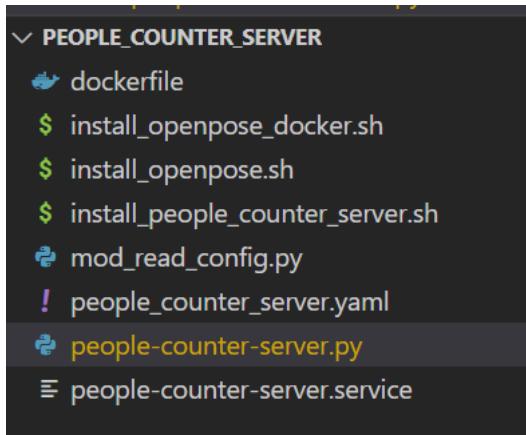


Abbildung 27 - Projektordner "PEOPLE_COUNTER_SERVER"

Der *OpenPose-Server* stellt das serverbasierte Gegenstück des People Counters dar. Vereinfacht kann er als zentrales Hirn dieses Sensors bezeichnet werden, da er die Möglichkeit bietet Bilddaten von verschiedenen People Counter Edge Devices entgegenzunehmen und zu verarbeiten.

Abbildung 27 zeigt den Projektordner „PEOPLE_COUNTER_SERVER“, mit allen nötigen Komponenten für den Betrieb dieses Dienstes.

Das Hauptprogramm für den OpenPose-Server ist die Datei *people-counter-server.py*, welche wie in Abbildung 28 verkürzt dargestellt zu sehen, die beiden REST-API Endpunkte */JsonPeopleCounting* (Abbildung 21 – Zeile 103) und */FilePeopleCounting* (Abbildung 21 – Zeile 124) über FLASK bereitstellt.

```

102     #Route für POST Request über /JsonPeopleCounting
103     @app.route('/JsonPeopleCounting',methods=['POST'])
104
105 > def json2image(): ...
122
123     #Route für POST Request über /FilePeopleCounting
124     @app.route('/FilePeopleCounting',methods=['POST'])
125
126 > def getPicture(): ...
144
145     #Starte FLASK Service auf port 8443
146     #Erstelle ein AdHoc selbstsigniertes Zertifikat für Verschlüsselte Kommunikation
147     #Horche auf allen verfügbaren IP-Adressen (0.0.0.0)
148     if __name__ == '__main__':
149         app.run(host="0.0.0.0", port=8443,ssl_context='adhoc')
```

Abbildung 28- *people-counter-server.py* - REST-API-EndPoints

Zudem beinhaltet der Projektordner noch BASH-Skripte für die Installation aller nötigen Komponenten und Bibliotheken, welche für den Betrieb des Dienstes mit OpenPose innerhalb eines Docker-Containers oder als kompilierte Variante benötigt werden.

1.4.5.2 MQTT – InfluxDB – Broker



Abbildung 29 - MQTT-InfluxDB-Broker

Zentrale Aufgabe des *MQTT-InfluxDB-Brokers* ist es, die gewonnenen Sensordaten per MQTT zu Empfangen und in eine Zeitreihendatenbank zu übertragen. Zudem wird hier das Arbeitsverhalten der erarbeiteten Lösung geregelt. Um diese Aufgabe kümmert sich das Python-Skript „`mqtt-to-influx-broker.py`“ als Hauptprogramm, welches sich im Projektordner „MQTT-INFLUX-BROKER“ befindet. Wie in Abbildung 29 zu sehen, kommen *Mosquitto* als MQTT-Broker und *InfluxDB* als Zeitreihendatenbank, welche auf dem Server als Dienste installiert und konfiguriert sind, zum Einsatz.

```
▽ MQTT-INFLUXDB-BROKER
  $ install_mqtt-to-influx-broker.sh
  ↗ mod_read_config.py
  ↗ mod_sendData.py
  ↗ mqtt-to-influx-broker.py
  ≡ mqtt-to-influx-broker.service
  🔒 my_ca.crt
  ! server_config_pattern.yaml
  ! server_config.yaml
```

Abbildung 30 - Projektordner "MQTT-INFLUXDB-BROKER"

Wie in Abbildung 30 zu sehen, befindet sich im Projektordner „MQTT-INFLUXDB-BROKER“ ebenfalls das Bash-Skript „`install_mqtt-to-influx-broker.sh`“, welche alle benötigten Komponenten und Dienste auf dem Server installiert.

Damit das Hauptprogramm seine Aufgabe erfüllen kann, muss es mittels einer Konfigurationsdatei im JSON oder YAML-Format konfiguriert werden. Wie in Abbildung 31 zu sehen, kann dabei jede nötige Serverkomponente des *MQTT-InfluxDB-Brokers* der gegebenen Umgebung frei konfiguriert werden.

Hierzu dienen innerhalb der Konfigurationsdatei die Punkte *general_config* (Abbildung 24 – Zeile 1-12), *mqtt_config* (Abbildung 24 – Zeile 15-48) und *influxdb_config* (Abbildung 24 – Zeile 51-57).

```
! server_config_pattern.yaml
1   #General Config of the Serverbehavior
2   <general_config:
3     start_hour: 6
4     end_hour: 22
5     influencer:
6       - testroom/sensor/people_counter/usage
7       - testroom/sensor/room/usage
8     default_workmode: 0
9   <switch_master :
10    | - testroom/sensor/people_counter/usage
11    switch_max_counter: 3
12    switch_back_mode : 1
13
14  #Config for MQTT-Server
15  <mqtt_config:
16    mqtt_server: 192.168.178.36
17    mqtt_port: 1883
18    mqtt_timeout: 60
19  > mqtt_subs: ...
20    mqtt_qos: 0
21    mqtt_tls_enable: false
22    mqtt_tls_port : 8888
23    mqtt_tls_ca_cert_file: my_ca.crt
24    #mqtt_tls_ca_cert_file: /etc/mosquitto/ca_certs/my_ca.cert
25    #for future use
26    #mqtt_tls_server_cert_file : /etc/mosquitto/certs/my_server.cert
27    #mqtt_tls_server_key_file : /etc/mosquitto/certs/my_server.key
28    mqtt_tls_user_name: Influencer
29    mqtt_tls_user_pass: Influencer
30
31  #Config for InfluxDb
32  <influxdb_config:
33    influxdb_server: 192.168.178.36
34    influxdb_port: 8086
35    influxdb_tls_enable: true
36    influxdb_api_token: edd1bc0c-99a0-4b73-9c99-0d9a263e6b8c
37    influx_bucket: sensor_data
38    influx_org: sedus
```

Abbildung 31 - YAML-Konfiguration

Dadurch das es sich bei *Mosquitto* und *InfluxDB* um eigenständige Serverdienste handelt, können diese auch dezentral in eigenen oder bereits existierenden Instanzen betrieben und genutzt werden.

Dies macht die erarbeitete Lösung in ihrem Einsatz sehr flexibel.

2 Umsetzung

2.1 Problem und Lösungsansatz



Abbildung 32- Agile Projektfläche mit se:lab (Quelle: sedus.com)

In den letzten 150 Jahren hat sich die Firma Sedus im Büromöbelbereich zu einer der zehn führenden Büromöbelmarken in Europa entwickelt. Neben der Entwicklung, Produktion und dem Vertrieb von eigenen Büromöbeln wie Stühlen, Tischen, Schränken und Raum-In-Raum Systemen bietet Sedus seinen Kunden auch Beratungs- und Planungsleistungen zu bestehenden Büroflächen an.

In Zeiten der Covid-19 Pandemie und der immer größeren Akzeptanz und Einführung von Home-Office in der Arbeitswelt, stehen Unternehmen nun vor der Herausforderung Ihre Büroflächen effizient umzuformen. Die Pandemie zeigt uns, dass Mitarbeiter nicht mehr zwingend einen fest zugeteilten Arbeitsplatz im Büro zur Verrichtung ihrer Tätigkeiten benötigen, sondern das viele dieser Arbeiten auch aus dem Home-Office heraus zufriedenstellend erledigt werden können.

Somit wird das klassische Büro für viele Arbeitnehmer immer unattraktiver und Unternehmen beginnen damit, das Büro zu einem Ort der Begegnung, Kollaboration und der agilen Arbeit umzuformen. Dank Konzepten wie z.B. Open Space und Desk Sharing entstehen in den Büros unter anderem auch immer mehr agile Projektflächen, in denen sich Mitarbeiter zu Brainstorming Sessions, Workshops oder kurzen, kreativen Besprechungen zusammenfinden können.

Agile Projektflächen definieren sich dadurch, dass sie nicht wie z.B. Konferenzräume starr und auf einen speziellen Zweck hin konzipiert sind, sondern sich spontan einem Zweck oder Bedürfnis anpassen können. Diesem Anspruch versucht Sedus mit der Möbelreihe se:lab gerecht zu werden, indem Möbel geschaffen wurden, die sich den Bedürfnissen und gewünschten Szenarien der Mitarbeiter für eine kollaborative Arbeit schnell und einfach anpassen lassen (Abbildung 32).

Jedoch ist es für Unternehmen nach der Einrichtung solcher agilen Flächen nur schwer festzustellen, ob diese und das dazu gehörende Mobiliar auch wirklich im gewünschten Ausmaß genutzt werden. Facility Manager eines Unternehmens können sich dahingehend meist nur auf eigene Beobachtungen oder die Aussage von Mitarbeitern verlassen und wissen daher meist nicht, ob solche Flächen zukünftig besser anders zu nutzen oder ausgebaut werden sollten.

Daher soll der Lösungsansatz verfolgt werden, die Nutzung solcher agilen Projektflächen und das dazugehörige Mobiliar sensorbasiert zu messen und die gewonnenen Daten für spätere Analysen abzuspeichern.

Hierzu werden die Eigenschaften und die Gegebenheiten eines agilen Projektraumes und dessen se:lab Möbel analysiert, dementsprechend passende Sensoren evaluiert und anschließend implementiert. Sensordaten sollen kabellos übertragen und in einem zentralen System abgespeichert werden können. Aus Datenschutzgründen darf kein Personenbezug möglich sein und die Kommunikation aller Komponenten soll weitestgehend verschlüsselt erfolgen.

Die erarbeitete Lösung dient als Proof-of-Concept, weswegen zunächst nur die reine Kommunikation und Interaktion aller Komponenten innerhalb eines Standortes berücksichtigt werden. Aspekte wie z.B. möglichst optimaler und geringer Stromverbrauch der Sensoren und anderer batteriebetriebener Komponenten werden zunächst nicht näher betrachtet, da dies eine eigene Entwicklung für sich darstellen würde.

2.2 Evaluierung von Hardware und Sensorik

Eine der größten Herausforderungen dieses Projektes stellt die Nutzungsmessung des agilen Projektraumes, sowie seiner agilen Projektmöbel dar. Dadurch dass Projektmöbel so konzipiert sind, dass sie keinen festen Platz im Raum haben und je nach Raumnutzung woanders stehen können, ist es zwingend notwendig, dass alle nötigen Komponenten, welche bei den Projektmöbeln zum Einsatz kommen, über eine eigene externe Stromquelle in Form einer Batterie oder eines Akkus verfügen, welche die Komponenten mit ausreichend Strom versorgen. Andere Komponenten, welche einen festen Platz im Projektraum haben, können hingegen über an Steckdosen angeschlossene Netzteile betrieben werden. Zudem müssen die Komponenten der Projektmöbel aufgrund der flexiblen Positionierung im Raum in der Lage sein, die gewonnenen Daten kabellos und möglichst energieeffizient zu übertragen. Aus Gründen der geplanten Energieeffizienz sollen alle Komponenten nur in vordefinierten Intervallen oder bei aktiver Nutzung Daten erheben und übertragen. Den Rest der Zeit sollen alle Komponenten in einem Ruhe-Modus verweilen, um Energie zu sparen.

2.2.1 Microcontroller – ESP32 NodeMCU

Um alle gewünschten Anforderungen abzudecken, kommt ein ESP32 – NodeMCU Developmentboard der Firma AZ-Delivery als Microcontroller zum Einsatz. Der Vorteil gegenüber eines anderen Microcontroller Boards wie z.B. einem Arduino Nano besteht darin, dass der ESP32 als System on a Chip (SoC) bereits Komponenten für die kabellose Übertragung von Daten, wie z.B. Bluetooth oder Wifi integriert hat. Zudem unterstützt der ESP32 sowohl diverse Deep Sleep Modi als auch die Möglichkeit über einen externen Impuls geweckt zu werden. Des Weiteren besitzt der ESP32 als weiteres Feature die Möglichkeit Variablen im Programmcode auch über einem Deep Sleep hinaus zu speichern, was sich im Laufe des Projekts als äußerst nützlich herausstellt.

Der ESP32 ist zudem auch über die Arduino IDE in der Programmiersprache C++ programmierbar und mit vielen Sensoren und dessen bestehenden Arduino Bibliotheken kompatibel, was die Entwicklung mit diesem Microcontroller sehr stark vereinfacht.

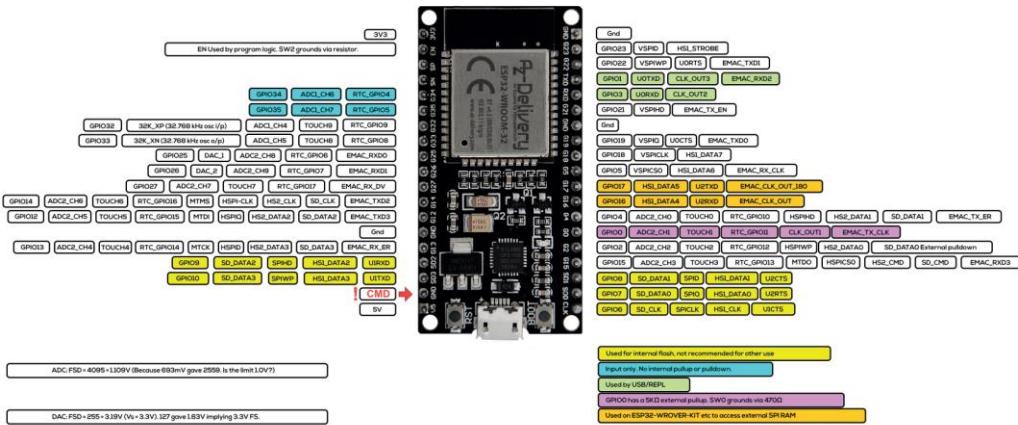


Abbildung 33 - ESP32-NodeMCU (Quelle:az-delivery.de)

Abbildung 26 zeigt das verwendete ESP32 Entwicklerboard in einem Pinout Diagramm.

2.2.2 Sensorik – Raum



Abbildung 34 - Sensorik für Raumsensor

Die Sensorik des Raumes soll neben der Nutzungserfassung des Projektraumes auch dessen Luftqualität messen können. Hierzu soll festgestellt werden können, wann ein Raum betreten wurde und wie sich die Luftqualität in Form des CO_2 -Gehalts, Temperatur und Luftfeuchtigkeit während seiner Nutzung verändert.

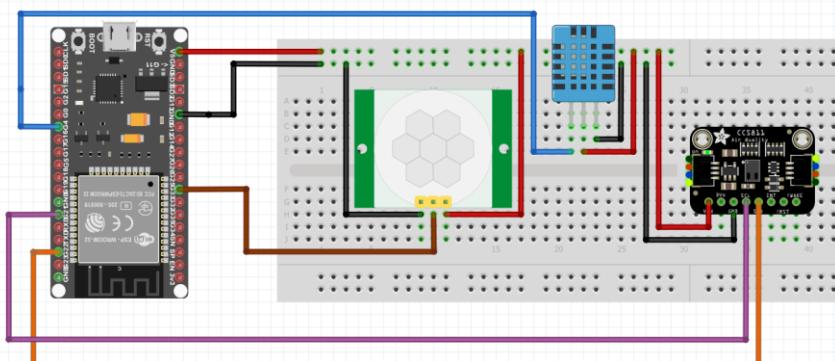
Abbildung 34 zeigt die für den Raumsensor verwendete Sensorik, dessen Einsatzwecke kurz erklärt werden.

Der *DHT11* ist ein Sensor für eine kalibrierte, digitale Signalausgabe der Temperatur und Luftfeuchtigkeit. Er erfasst, misst und meldet regelmäßig die relative Luftfeuchtigkeit und Temperatur der Luft. Die relative Luftfeuchtigkeit, ausgedrückt in Prozent, ist das Verhältnis der tatsächlichen Feuchtigkeit in der Luft zur höchsten Feuchtigkeitsmenge, die bei dieser Temperatur gehalten werden kann.

Je wärmer die Luft ist, desto mehr Feuchtigkeit kann sie halten, so ändert sich die relative Feuchte mit Temperaturschwankungen [50]. Der Sensor hat eine Genauigkeit von $\pm 2^{\circ}\text{C}$ in der Temperatur und $\pm 5\%$ bei der Luftfeuchtigkeit. Was diesen Sensor nicht sehr genau aber preisgünstig macht. Für einen ersten Proof-of-Concept ist dies jedoch mehr als ausreichend.

Zur Ermittlung, ob jemand den Projektraum betritt oder verlässt, kommt ein Passiv-Infrarot-Melder (PIR) vom Typ *HC-SR501* zum Einsatz. Passiv-Infrarot-Melder reagieren auf Veränderungen der auftreffenden Infrarot-Wärmestrahlung. Sie besitzen ein Linsensystem, das einen Erfassungsbereich definiert. Um Bewegungen erkennen zu können, müssen die bewegten Objekte selbst Wärme abstrahlen, wie es z.B. bei Menschen der Fall ist [51]. Der *HC-SR501* verfügt über einen Erfassungsbereich von 110° und kann auf Wärmeveränderungen auf bis zu 7m Entfernung reagieren [52]. Dadurch dass der Sensor einen Impuls bei erkannter Wärmeveränderung liefert, eignet er sich auch als Trigger Sensor, welcher den Raumsensor aus seinem Ruhemodus aufwecken kann.

Einen weiteren wichtigen Aspekt der Arbeitsqualität eines Raumes spiegelt die Luftqualität wider. Neben der Temperatur und der Luftfeuchtigkeit ist auch der CO_2 -Gehalt in der Luft ein wichtiger Indikator für eine gesunde und effektive Arbeitsumgebung. Bereits vor über 130 Jahren hatte der deutsche Forscher Max Pettenkofer den Kohlendioxid-Gehalt der Luft als Maßstab für die Qualität der Raumluft erkannt. Für das Befinden des Menschen ist der CO_2 -Gehalt der Luft also von großer Bedeutung. Steigt die Konzentration von Kohlendioxid in einem Zimmer an, werden wir müde und unsere Konzentration leidet [53]. Daher kommt für die Messung des CO_2 -Gehalts in der Luft ein *CCS811-Luftqualitätssensor* zum Einsatz. Dieser Sensor kann die Konzentration von **eCO₂** (äquivalent berechnetes Kohlendioxid) in einem Bereich zwischen 400 und 8192 Teilen pro Million (ppm) und die Konzentration von **TVOC** (Total Volatile Organic Compound) in einem Bereich zwischen 0 und 1187 Teilen pro Milliarde (ppb_{sp}) in der Luft messen [54].



fritzing

Abbildung 35 - Schaltbild des Raumsensors

Abbildung 35 zeigt ein vereinfachtes Schaltbild der verwendeten Komponenten des Raumsensors. Wie zu sehen, werden alle Komponenten über den ESP32 angesteuert und mit der nötigen Betriebsspannung versorgt. Da der Raumsensor einen festen Platz im Projektraum hat, welcher nicht abweicht, wird der ESP32 über ein an einer Steckdose angeschlossenes Netzteil mit Strom versorgt.

2.2.3 Sensorik – se:lab High Desk



Abbildung 36 - se:lab High Desk (Quelle: sedus.com)

Der se:lab High Desk vereint alle Vorteile des modernen Hochtischkonzeptes in sich. Er ist ideal für agile Teamarbeit oder als Treffpunkt für Kommunikation geeignet. Dabei begünstigt er mit einer Höhe von 900 mm spontane Haltungswechsel, was sich positiv auf Gesundheit und Kreativität auswirken kann. Dank der Ausstattung mit Rollen kann der se:lab High Desk unkompliziert umgestellt werden [55]. Dadurch dass dieser Tisch sowohl im Stehen als auch im Sitzen genutzt werden kann, muss die genutzte Sensorik in der Lage sein eine Nutzung nur dann zu erkennen, wenn auch eine Person aktiv an diesem Tisch arbeitet. Hierzu wird der Ansatz verfolgt, dass ein Tisch nur dann genutzt wird, wenn eine Person sich maximal 70 cm vom Sensor, welcher mittig an der unteren Tischplatte in einem Winkel von etwa 30° angebracht ist, befindet.

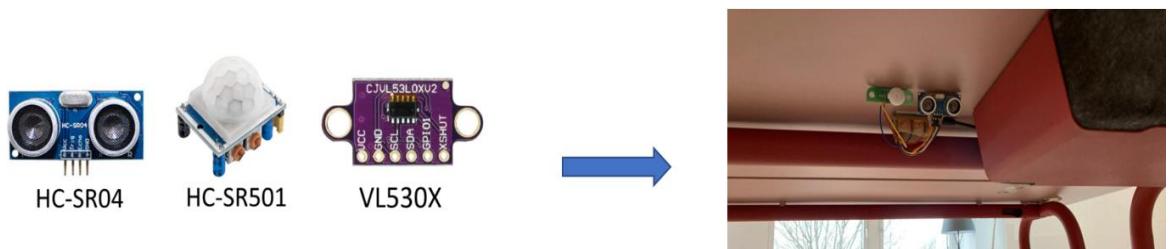


Abbildung 37 - Ultraschall Distanz Sensor (HC-SR04), PIR-Sensor (HC-SR501), LiDAR Distanz Sensor (VL530X)

Abbildung 37 zeigt hierbei drei Sensoren, die sich für die Nutzungsmessung des se:lab High Desk auf Grund der beschriebenen Anforderungen eignen. Der VL530X und der HC-SR04 kommen jeweils als eigener Distanzsensor zum Einsatz.

Der HC-SR04 ist ein kostengünstiger Distanz Sensor, welcher auf Basis von Ultraschall Entferungen messen kann. Hierzu sendet der Sensor ein Ultraschallsignal mit einer Frequenz von 40 kHz in den Raum, welches von Objekten zurückgeworfen und vom Sensor erfasst wird.

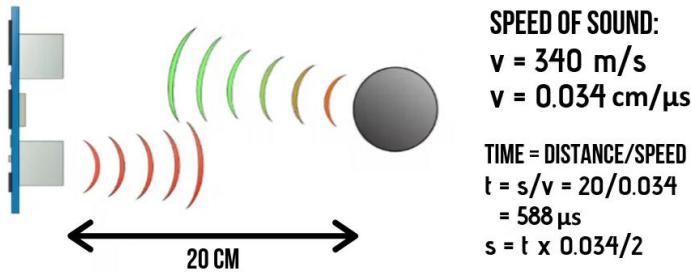


Abbildung 38-Berechnung der Distanz durch Ultraschall (Quelle: create.arduino.cc)

Abbildung 38 beschreibt hierbei die nötige Mathematik hinter dieser Art der Distanzmessung. Somit ergibt sich in diesem Beispiel, dass wenn sich der Schall mit einer Geschwindigkeit von 340 m/s oder auch 0.034 cm/μs [56] fortbewegt, dieser für eine Distanz von 20 cm etwa 588 μs benötigt. Dadurch das der Schall jedoch einen Hin- und Rückweg hat, muss diese Distanz anschließend halbiert werden. Somit ergibt sich in diesem Beispiel eine Distanz zum Objekt von etwa 10 cm.

Ein ähnliches Prinzip verfolgt auch der LiDAR (Light Detection and Ranging) Sensor VL530X. Jedoch verwendet dieser Sensor zentrierte Laser Lichtimpulse, um anhand der vergangenen Zeit einer Lichtreflektion die Distanz zu einem Objekt zu ermitteln [57].

Ein Distanz Sensor kann jedoch nicht unterscheiden, ob es sich bei einem Objekt, welches unter 70 cm entfernt ist, um eine Wand, ein Stuhl oder einen Menschen handelt. Daher kommt hier noch ein PIR-Sensor vom Typ HC-SR501 zum Einsatz.

Wie bereits beim Raum-Sensor beschrieben regiert ein PIR-Sensor auf Veränderungen der auftreffenden Infrarot-Wärmestrahlung, welche z.B. von Menschen abgegeben werden. Somit eignet sich dieser Sensor sowohl zur Bestimmung einer menschlichen Präsenz, als auch als Trigger zum Aufwecken des ESP32-Entwicklerboards.

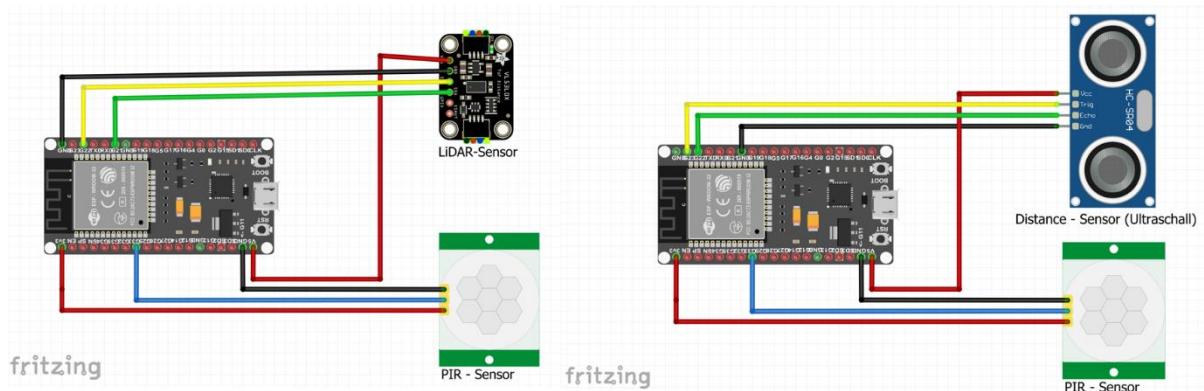


Abbildung 39 - Schaltbild High-Desk-Sensoren

Abbildung 39 zeigt ein vereinfachtes Schaltbild der verwendeten Komponenten der beiden High-Desk-Sensoren. Wie zu sehen, werden alle Komponenten über den ESP32 angesteuert und mit der nötigen Betriebsspannung versorgt. Auf Grund der Flexibilität dieses Tisches muss der ESP32 per Akku mit Strom versorgt werden.

2.2.4 Sensorik – se:lab Board



Abbildung 40- se:lab Board (Quelle: sedus.com)

se:lab Boards sind leichte, tragbare Whiteboards, die sich überallhin mitnehmen lassen und so agiles Arbeiten fördern. Sie sind beidseitig beschreibbar, magnetisch und optimal für Haftnotizen geeignet [58].

Durch die leichte Bauweise des se:lab Boards ergibt es sich, dass die Statik des Gestells nicht fix und starr ist, sondern einen gewissen Spielraum für Bewegungen in Form von Vibrationen / Schwingungen bietet. Wird das se:lab Board benutzt, beginnt das Gestell damit leicht vor und zurück zu schwingen. Eine stärkere Wackelbewegung des Gestells tritt nur bei aktiver Benutzung in Form von schreiben auf einem Whiteboard oder durch Fortbewegung des Gestells auf. Erschütterungen wie z.B. Stampfen oder Springen neben dem Gestell reichen dagegen i.d.R nicht aus um das se:lab Board ausreichend in Schwingung zu versetzen. Daher dient zur Nutzungsmessung dieses Möbelstücks der Ansatz, dass eine Nutzung immer dann vorliegt, wenn das Gestell sich in einem ausreichend schwingenden Zustand befindet.

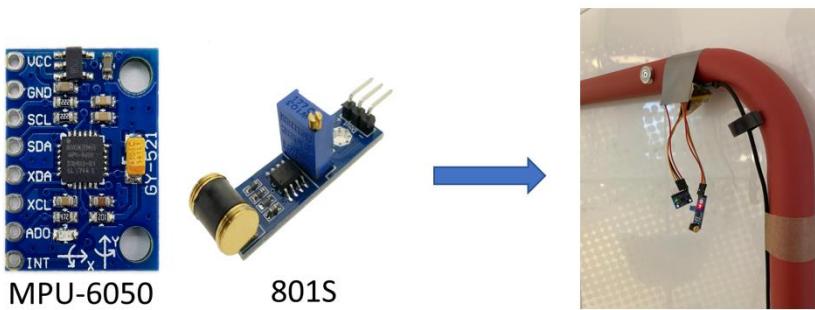


Abbildung 41 - v.l. Gyrosensor (MPU-6050) und Vibrationssensor (801S)

Zum Messen dieser Schwingungen kommen die in Abbildung 41 dargestellten Sensoren zum Einsatz. Beide Sensoren sind an der Oberseite des se:lab Board freischwingend angebracht, was diese bei einer Benutzung in Schwingung versetzt und nachpendeln lässt.

Beim *MPU6050* handelt es sich um ein 3-Achsen Beschleunigungssensor und 3-Achsen Gyroskop. Darüber hinaus kann der *MPU6050* die Temperatur messen. Der *MPU-6050* nutzt für seine Messungen „Micro-Electric-Mechanical Systems“ (MEMS) [59].

Im Fall von Gyroskopen sind das beweglich aufgehängte Körper, die bei Beschleunigung ihre Lage gegenüber einem festen Rahmen verändern. Die Abstandsänderung führt zu einer Änderung der Kapazität (*Abbildung 42*) [60].

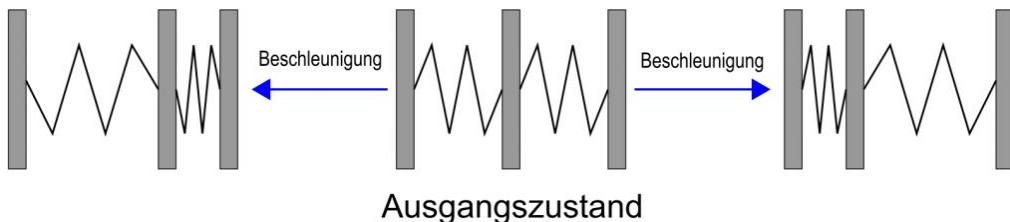


Abbildung 42 - Messprinzip MPU6050 (Quelle: wolles-elektronikkiste.de)

Der 801S ist ein empfindlicher Vibrationssensor, welcher mit Hilfe des Piezoelektrischen Effekts [61] feinste Erschütterungen wahrnehmen und messen kann [62]. Diese Eigenschaft ermöglicht es auch feinste Bewegungen zu messen, die beim Benutzen eines Whiteboards am se:lab Board entstehen. Zudem eignet sich dieser Sensor als Trigger zum Aufwecken des ESP32-Entwicklerboards.

Dadurch das der *801S* auch feinste Erschütterungen messen kann, die in einigen Gebäuden z.B. bei vorbeifahrenden Zügen oder LKWs entstehen können, wird zur Nutzungsbestimmung ebenfalls der Gyrosensor des *MPU6050* genutzt.

Eine Nutzung des se:lab Boards wird somit nur erkannt, wenn beide Sensoren einen vordefinierten Richtwert überschreiten. Nur wenn bei beiden Sensoren eindeutig eine Nutzung erkannt wurde, gilt das se:lab Board als benutzt.

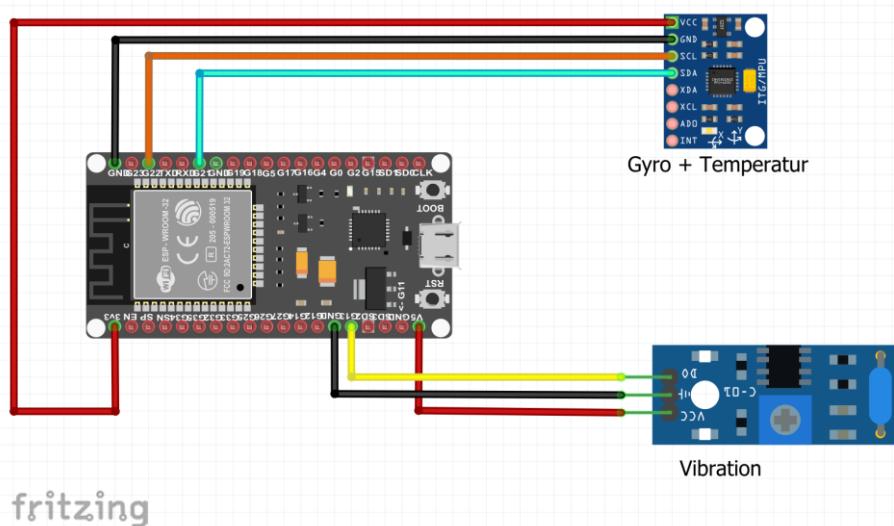


Abbildung 43 - Schaltbild se:lab Board Sensor

Abbildung 43 zeigt ein vereinfachtes Schaltbild der verwendeten Komponenten des se:lab Board Sensors. Wie zu sehen, werden alle Komponenten über den ESP32 angesteuert und mit der nötigen Betriebsspannung versorgt. Auf Grund der Flexibilität des se:lab Boards muss der ESP32 per Akku mit Strom versorgt werden.

2.2.5 Sensorik – se:lab Hopper



Abbildung 44 - se:lab Hopper (Quelle: sedus.com)

Der se:lab Hopper ist die ideale Ergänzung für Workshops, Projektmeetings oder agile Worksettings, denn dieses Sitzmöbel fördert Spontanität und Interaktion mit einer angenehmen Höhe von 690 mm. Der Sitzbock ist im Gebrauch mobil und wendig. Wird er nicht benötigt, lässt er sich platzsparend stapeln oder ineinander stellen [63].

Dadurch das es sich beim se:lab Hopper um ein Sitzmöbel handelt, reicht es für die Nutzungsbestimmung aus, festzustellen ob eine Person auf dem Möbel sitzt oder nicht. Der verwendete Sensor darf jedoch nicht zu fein sein, da eine Nutzung nur erfolgen soll, wenn eine Person und nicht beispielsweise eine Jacke oder eine Tasche auf dem Sitzbock aufliegt. Zudem soll der verwendete Sensor möglichst nicht beim Sitzen spürbar sein, weswegen der Ansatz von stoffbasierten Drucksensoren verfolgt wird.

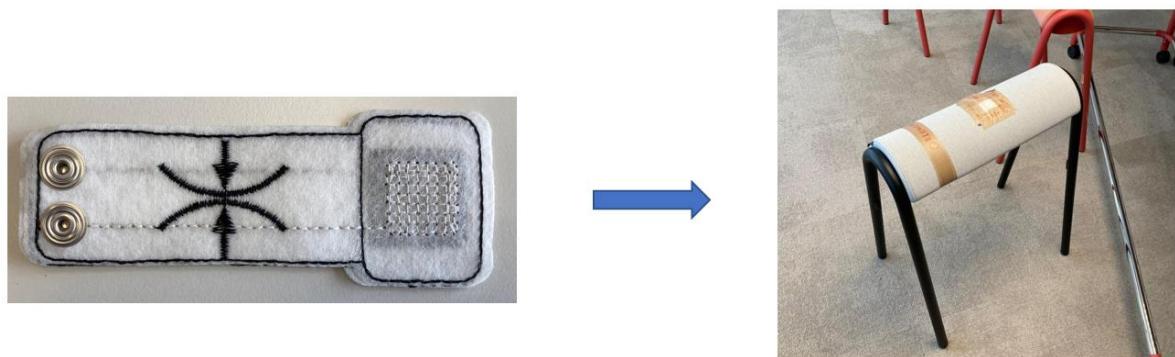


Abbildung 45 - Stoffbasierter Drucksensor von Wearic

Abbildung 45 zeigt einen stoffbasierten Drucksensor aus dem Smart Textiles Kit der Firma Wearic [64]. Durch Sitzen auf dem Sensor (wie in Abbildung 37 zu sehen) ändert sich der Widerstand des Sensorkopfes. Je nach ausgeübtem Druck auf den Sensorkopf entsteht hier ein Widerstand zwischen 100 Ohm bis zu 100 Kilo-Ohm, welcher gemessen, ausgewertet und interpretiert werden kann.

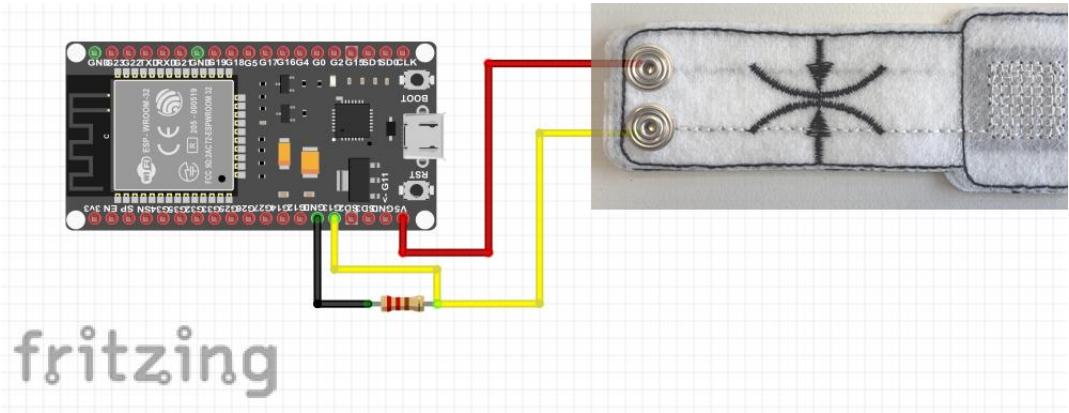


Abbildung 46 - Schaltbild se:lab Hopper Sensor

Abbildung 46 zeigt ein vereinfachtes Schaltbild der verwendeten Komponenten des se:lab Hopper Sensors. Wie zu sehen, werden alle Komponenten über den ESP32 angesteuert und mit der nötigen Betriebsspannung versorgt. Da der Drucksensor aus einem vorgefertigten Test-Kit stammt, muss dieser mit Hilfe des „WEARIC - EXPANSION BOARD“ [65] Datenblattes auf den ESP32 abgeändert werden. Auf Grund der Flexibilität des se:lab Hoppers muss der ESP32 per Akku mit Strom versorgt werden.

2.2.6 Edge-Devices – Raspberry Pi 3B+



Abbildung 47 - Raspberry Pi 3B+ (Quelle: reichelt.de)

Die Anforderungen an ein Edge-Device sind, dass dieses in der Lage sein muss, kabellos Sensordaten zu empfangen und über eine IP-fähige Schnittstelle an Serversysteme in angemessener Geschwindigkeit und Formatierung, weiterzuleiten. Zudem muss ein Edge Device auch in der Lage sein eine USB-Kamera anzusteuern um Bildaufnahmen zu erstellen. Optional soll auch eine Stromversorgung per Power-over-Ethernet (PoE) möglich sein.

Um die genannten Anforderungen optimal abdecken zu können, kommen Single Board Computer des Typs „Raspberry Pi 3B+“ zum Einsatz.

Specification

The Raspberry Pi 3 Model B+ is the final revision in the Raspberry Pi 3 range.

Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
1GB LPDDR2 SDRAM
2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
Extended 40-pin GPIO header
Full-size HDMI
4 USB 2.0 ports
CSI camera port for connecting a Raspberry Pi camera
DSI display port for connecting a Raspberry Pi touchscreen display
4-pole stereo output and composite video port
Micro SD port for loading your operating system and storing data
5V/2.5A DC power input
Power-over-Ethernet (PoE) support (requires separate PoE HAT)

Abbildung 48 - Raspberry Pi 3B+ Spezifikationen (Quelle: raspberrypi.com)

Wie in den Spezifikationen des Raspberry Pi 3B+ (siehe Abbildung 48) zu sehen, verfügt dieser über alle geforderten Eigenschaften, die ein Edge Device mitbringen sollte. Dadurch das dieser Single Board Computer mit einer angepassten Debian Linux-Distribution betrieben werden kann, können eine Vielzahl von Tools, Programmiersprachen und Bibliotheken aus dem Linux Umfeld zum Erfüllen der gewünschten Ziele genutzt werden.

2.3 Kommunikation zwischen ESP32-Boards und Edge Devices

Dadurch das die verbaute Sensorik in agilen Projektmöbeln eingesetzt wird, ist eine kabellose Datenübertragung zwingend notwendig. Ein Blick in die technischen Daten (Abbildung 49) des ESP32-NodeMCU veranschaulicht, dass dieser sowohl über Wi-Fi, als auch über eine Bluetooth 4.2 Schnittstelle, welche auch BLE kann, verfügt.

Technische Daten

Stromversorgungsspannung (USB)	5V
Eingangs-/Ausgangsspannung	3.3V
Benötigter Betriebsstrom	min. 500mA
SoC	ESP32-WROOM 32
Taktfrequenzbereich	80MHz / 240MHz
RAM	512kB
Externer Flash-Speicher	4MB
I/O Pins	34
Schnittstellen	SPI, I2C, I2S, CAN, UART
Wi-Fi Protokolle	802.11 b/g/n (802.11n bis zu 150 Mbps)
Wi-Fi Frequenz	2.4 GHz - 2.5 GHz
Bluetooth	V4.2 - BLE und Classic Bluetooth
Drahtlose Antenne	PCB
Abmessungen	56x28x13mm

Abbildung 49 - Technische Daten ESP32 (Quelle: az-delivery.de)

Im Vergleich der beiden Technologien, bietet zwar Wi-Fi eine höhere Reichweite und Datenrate, jedoch ist dafür auch der Stromverbrauch um bis zu zehn Mal höher gegenüber dem Bluetooth Low Energy Standard [66]. Dadurch das sich die Sensorik innerhalb eines Projektraumes mit etwa 36 qm befindet, mit den Sensordaten nur geringe Informationsmengen ausgetauscht werden müssen und zudem ein ESP32 Board möglichst lange mit einem Akku betrieben werden soll, fällt hierbei die Entscheidung auf eine Datenübertragung über Bluetooth Low Energy.

2.3.1 Messen und Senden von Sensordaten

Wie bereits beschrieben erheben die ESP32-Boards mit Hilfe der angeschlossenen Sensoren Daten und versenden diese als Advertising Paket (Broadcast) per BLE, sodass diese von den Edge-Devices aufgenommen und weiterverarbeitet werden können.

```

281 // Create the BLE Device
282 BLEDevice::init("SEDUS-TABLE");
283 // Create the BLE Server
284 pServer = BLEDevice::createServer();
285
286 //for Serial-Log and identification of ESP-Board over BLE
287 Serial.print("MY BLE-ADRESS IS: ");
288 printBLEAddress();
289
290 pService = pServer->createService(SERVICE_UUID);
291
292 pCharacteristic = pService->createCharacteristic(
293                                         CHARACTERISTIC_UUID,
294                                         BLECharacteristic::PROPERTY_READ | 
295                                         BLECharacteristic::PROPERTY_WRITE
296                                         );
297
298 pServer->setCallbacks(new MyServerCallbacks());
299 pCharacteristic->setCallbacks(new MyCallbacks());
300
301 pCharacteristic->setValue("Start");
302
303 pAdvertising = pServer->getAdvertising();
304
305 //Get Sensordata and set it in Eddystone Beacon
306 setBeacon();
307
308 //start BLE-Services
309 pService->start();
310     // Start advertising
311 pAdvertising->start();
312 Serial.println("Advertising started...");
313 //advertise for 7.5 Seconds
314 delay(7500);

```

Abbildung 50 - Implementierung von GAP und GATT in Arduino Sketch

Abbildung 50 zeigt hierbei einen Auszug der Implementierung dieses Vorgangs. In der Zeile 284 wird zunächst definiert, dass es sich bei dem ESP32-Board um einen BLE-Server handeln soll, welcher Dienste über GAT bereitstellt. In den Zeilen 290 – 301 erfolgt die Definition dieser Dienste samt Callbacks, welche bei Nutzung der Dienste ausgeführt werden sollen. Die Services und Callbacks ermöglichen hierbei die Manipulation der Working Modes durch die Edge Devices. In den Zeilen 303-314 werden die Sensordaten schlussendlich erfasst und per BLE-Advertising als Broadcast versendet.

```

163 void setBeacon() {
164
165     char beacon_data[22];
166     uint16_t beconUUID = 0xFEEA;
167
168     BLEAdvertisementData oAdvertisementData = BLEAdvertisementData();
169
170     oAdvertisementData.setFlags(0x06); // GENERAL_DISC_MODE 0x02 | BR_EDR_NOT_SUPPORTED 0x04
171     oAdvertisementData.setCompleteServices(BLEUUID(beconUUID));
172
173     //Get Sensor-Data
174     bool trigger_status = get_trigger_status();
175     bool used_status = false;
176     char sensor_type = 'T'; //TABLE
177     char temp_prefix = '+';
178     bool pir_status = get_pir();
179     int distance = get_distance();
180     int perc_battery = get_battery();
181
182     if ( ((distance <= 70) && (distance > 0) ) && (pir_status == true)) {
183         used_status = true;
184     }
185
186     //Set Eddystone Beacon
187     beacon_data[0] = 0x20; // Eddystone Frame Type (Unencrypted Eddystone-TLM)
188     beacon_data[1] = 0x00; // TLM version
189     beacon_data[2] = working_mode; //
190     beacon_data[3] = sensor_type; // 'T' for Table
191     beacon_data[4] = used_status; //
192     beacon_data[5] = trigger_status;
193     beacon_data[6] = false; //
194     beacon_data[7] = false; //
195     beacon_data[8] = distance;
196     beacon_data[9] = false;
197     beacon_data[10] = false;
198     beacon_data[11] = false;
199     beacon_data[12] = false;
200     beacon_data[13] = perc_battery;
201
202     oAdvertisementData.setServiceData(BLEUUID(beconUUID), std::string(beacon_data, 14));
203
204     pAdvertising->setScanResponseData(oAdvertisementData);
205

```

Abbildung 51 - Aufbau des Eddystone Beacons in Arduino Sketch

Abbildung 51 zeigt die Implementierung der Methode `setBeacon()` aus Abbildung 50 – Zeile 306. Innerhalb von `setBeacon()` werden unter anderem weitere Methoden aufgerufen, welche mit Hilfe der verbauten Sensorik Messwerte erheben. Anhand dieser gewonnenen Daten wird zudem nach einem vordefiniertem Regelwerk eine Nutzung ermittelt und diese Ergebnisse zu einem Eddystone Beacon zusammengefügt, welcher anschließend per BLE als Advertising Paket versendet wird (Abbildung 50 – Zeile 311). Die Zeilen 174 – 204 zeigen beispielsweise, wie die Daten für den Tischsensor zunächst gesammelt und anschließend an definierter Stelle in ein `beacon_data-Char-Array` eingefügt werden.

Tabelle 1 veranschaulicht hierbei die Definition der verwendeten Bytes in Slots pro Sensor innerhalb des Eddystone Beacons, welcher in Abbildung 51 in den Zeilen 186-200 erstellt wird.

	Raumsensor	Whiteboard	CHAIR	Table	ByteSlots
Bezeichnung					
0 Eddystone Frame	Eddystone Frame	Eddystone Frame	Eddystone Frame	Eddystone Frame	Beacon Slot
1 TLM Version	TLM Version	TLM Version	TLM Version	TLM Version	
2 WORKINGMODE	WORKINGMODE	WORKINGMODE	WORKINGMODE	WORKINGMODE	
3 SENSORTYPE	SENSORTYPE	SENSORTYPE	SENSORTYPE	SENSORTYPE	Work Slot
4 USAGE	USAGE	USAGE	USAGE	USAGE	
5 PIR	Vibration	Seatresistance	PIR		Trigger slot
6 Temperature-Prefix	Temperature-Prefix	not used	not used		
7 Temperature (Celsius)	Temperature (Celsius)	not used	not used		Temperature slot
8 Humidity (%)	Vibration Detail - Byte 1	Seat Detail - Byte 1	Distance CM		
9 Airquality (CO2) Byte 1	Vibration Detail - Byte 0	Seat Detail - Byte 0	not used		
10 Airquality (CO2) Byte 0	Gyro x	not used	not used		Diverse Slot
11 not used	Gyro y	not used	not used		
12 not used	Gyro z	not used	not used		
13 Battery (%)	Battery (%)	Battery (%)	Battery (%)	Battery (%)	Battery Slot

Tabelle 1 - Definition Eddystone Beacon Bytes

2.3.2 Empfangen und Weiterleiten von Sensordaten

Um die per BLE-Advertising versendeten Sensordaten zu empfangen, auszuwerten und an einen MQTT-Broker weiterleiten zu können, kommt ein Raspberry Pi 3B+ zum Einsatz, welcher innerhalb der erarbeiteten Lösung als BLE-to-MQTT-Gateway fungiert.

```

39     #async function for BLE Advertising Broadcast Packages
40     async def main():
41         devices = await BleakScanner.discover()
42         for device in devices:
43             #Device name should be something with Sedus
44             if ("SEDUS" in device.name) or ("Sedus" in device.name): ...
45             else:
46                 try:
47                     #Scan for Servicedata with 0000feaa-0000-1000-8000-00805f9b34fb (Eddystone Service UUID)
48                     if (device.details['props']['ServiceData'][0]['0000feaa-0000-1000-8000-00805f9b34fb']):
49                         log_time = datetime.fromtimestamp(time()).strftime('%Y-%m-%d %H:%M:%S')
50                         print("{} - {}".format(log_time,device))
51                         try:
52                             if MQTT_SECURE == False:
53                                 send_mqtt(MQTT_SERVER,device,port=MQTT_PORT,timeout=MQTT_TIMEOUT,mqtt_room=MQTT_ROOM_TOPIC)
54                             else:
55                                 send_mqtt_secure(MQTT_SERVER, ...
56                             except:
57                                 print("Error while transacting Data")
58                         except:
59                             pass
60
61             #Run forever as a Service
62             while True:
63                 try:
64                     asyncio.run(main())
65                 except:
66                     log_time = datetime.fromtimestamp(time()).strftime('%Y-%m-%d %H:%M:%S')
67                     print("{} - Error with BLE-Interface occured".format(log_time))
68                     sleep(5.0)
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91

```

Abbildung 52 - Main-loop des BLE-to-MQTT Gateways

Hierzu dient das Python-Skript *ble-mqtt-gateway.py*, welches innerhalb einer *While True*-Schleife permanent die Umgebung nach Bluetooth Datenpaketen mit dem Wort *Sedus* im Gerätenamen oder der Eddystone Beacon UUID '0000feaa-0000-1000-8000-00805f9b34fb' scannt.

Abbildung 53- Daten aus Eddystone Beacon für se:lab Board (Whiteboard)

	Hex	ASCII	Integer	
Eddystone Frame	0x20	SPACE		beacon_data[0] = 0x20; // Eddystone Frame Type (Unencrypted Eddystone-TLM)
TLM Version	0x00		0	beacon_data[1] = 0x00; // TLM version
WORKINGMODE	0x02		2	beacon_data[2] = working_mode; //
SENSORTYPE	0x57	W	87	beacon_data[3] = sensor_type; // 'W' for Whiteboard
USAGE	0x00		0	beacon_data[4] = used_status; //
Vibration	0x00		0	beacon_data[5] = trigger_status;
Temperature-Prefix	0x2b	+	43	beacon_data[6] = temp_prefix;
Temperature (Celsius)	0x1b		27	beacon_data[7] = temperature; //
Vibration Detail - Byte 1	0x00		0	beacon_data[8] = (vibration_detail >> 8) & 0xFF; //bigger_byte;
Vibration Detail - Byte 0	0x00		0	beacon_data[9] = (vibration_detail >> 0) & 0xFF; //lower_byte;
Gyro x	0x00		0	beacon_data[10] = gyro_x;
Gyro y	0x00		0	beacon_data[11] = gyro_y;
Gyro z	0x00		0	beacon_data[12] = gyro_z;
Battery (%)	0x65	e	101	beacon_data[13] = perc_battery;

Abbildung 54 - Gegenüberstellung Beacon Bytes zu Daten aus Abb.51

Abbildung 53 zeigt beispielhaft ein vom BLE-to-MQTT-Gateway empfangenes Advertising Paket eines ESP32-Boards, welches an einem Whiteboard angebracht ist. Zu sehen ist, dass die empfangenen Daten von Python als Bytarray interpretiert werden. Hierbei kann es vorkommen das ein Byte einem darstellbaren ASCII-Zeichen entspricht, weswegen zum besseren Verständnis die Abbildung 54 dienen soll.

```

14     #unsecured MQTT Communication
15     def send_mqtt_data(mqtt_server,sensor_obj,port=1883,timeout=60,mqtt_room='testroom'):
16         #MQTT_SERVER = '192.168.178.36'
17
18         client = mqtt.Client("")
19         client.connect(mqtt_server,port,timeout)
20         bytedata = sensor_obj.details['props']['ServiceData']['0000feaa-0000-1000-8000-00805f9b34fb']
21         sensorID = str(sensor_obj.address).replace(':', '')
22         public_wait = 0.2
23         mqtt_publish_data(mqtt_server,port,client,sensor_obj,bytedata,sensorID,public_wait,mqtt_room_topic=mqtt_room)

```

Abbildung 55 - *send_mqtt_data*

Sobald der BLE-to-MQTT-Broker ein entsprechendes Datenpaket empfängt, wird die Methode *send_mqtt_data* bei unverschlüsselter MQTT-Kommunikation, oder *send_mqtt_data_secure* bei verschlüsselter MQTT-Kommunikation, aufgerufen. Hierbei wird zunächst eine Verbindung zum MQTT Broker aufgebaut (*Abbildung 55 – Zeile 18-19*), so wie die MAC-Adresse des ESP32-Boards ausgelesen (*Abbildung 55 – Zeile 21*), welche zur eindeutigen Identifizierung des Edge Devices dienen soll. Dies ist unter anderem dafür nötig um bei einer Erweiterung der erarbeiteten Lösung nicht den Überblick über die eingesetzten Edge Devices zu verlieren. Die *public_wait*-Variable dient zur künstlichen Verlangsamung des Skriptes, das es sonst beim Publishen der Sensordaten dazu kommen kann, dass diese vom MQTT-Broker nicht verarbeitet werden können. Das Publishing der Sensordaten erfolgt über die Methode *mqtt_publish_data*.

```

193     #Whiteboard Rack sensor - Whiteboard
194     if chr(bytedata[3]) == 'W':
195         print("Whiteboard Rack")
196         print("Modus: ",bytedata[2])
197
198         client.publish("{} /sensor/se_lab_board_{} /sensor_mode".format(mqtt_room_topic,sensorID),bytedata[2])
199         sleep(public_wait)
200
201         #Ask current working_mode via MQTT
202         working_mode = get_current_workingmode(mqtt_server,mqtt_port)
203
204         #If Working_mode is different then current mode of sensor it have to be switched
205         if (bytedata[2] != working_mode):
206             print("Change working mode form {} to {}".format(bytedata[2],working_mode))
207             set_sensor_working_mode(sensor_obj.address,working_mode)
208
209         #Used-Flag
210         print("Used: ",bytedata[4])
211         client.publish("{} /sensor/se_lab_board_{} /usage".format(mqtt_room_topic,sensorID),bytedata[4])
212         sleep(public_wait)

```

Abbildung 56 - Übersetzen und Senden von Beacon Bytes in MQTT-Topics

Abbildung 56 zeigt einen Auszug aus der Methode *mqtt-publish-data*. Innerhalb dieser Methode wird anhand des empfangenen *Bytarrays* überprüft um welche Art von Sensordaten es sich handelt (*Abbildung 56- Zeile 194*). In diesem Beispiel handelt es sich um die bereits beschrieben Sensordaten eines Whiteboards. Sobald der Sensortyp identifiziert ist, wird das empfangene *Bytarray* entsprechend den vordefinierten Positionen ausgelesenen, verarbeitet und als entsprechendes MQTT-Topic an den MQTT-Broker gepublished. In *Abbildung 56 – Zeile 198* würde z.B. der Working-Mode eines se:lab Boards im Testroom über das Topic *testroom/sensor/se_lab_board_0815/sensormode* mit dem Wert in *bytedata[2]* an den MQTT-Broker übertragen werden. Des Weiteren erfolgt hier auch der Wechsel des Working Modes eines entsprechenden ESP32-Boards. Hierzu wird zunächst in *Zeile 202* der aktuelle Working Mode vom MQTT-Broker per MQTT-Subscribe abgefragt und anschließend in *Zeile 205* mit dem Working-Mode, welches im *Bytarray* steht, verglichen.

```

4 def set_sensor_working_mode(ble_address, mode):
5     try:
6         gatt_device = pexpect.spawn("gatttool -I")
7         gatt_device.sendline("connect {}".format(ble_address))
8         #gatt_device.expect("Connection successful", timeout=5)
9         sleep(0.5)
10        #SENDs W(0x57),R(0x52),T(0x54) and WORKINGMODE{0,1,2}
11        gatt_device.sendline("char-write-req 0x002a 5752540{}".format(mode))
12        sleep(0.5)
13        gatt_device.sendline("disconnect")
14        sleep(0.5)
15        gatt_device.sendline("exit")
16        gatt_device.kill(0)
17    except:
18        print("error writing on {}".format(ble_address))
19        pass

```

Abbildung 57 - Überschreiben des aktuellen working_mode per BLE

Falls der Working Mode des MQTT-Brokers vom Eigenen abweicht, wird eine Änderung per BLE durchgeführt. Hierzu dient die Methode *set-sensor-working-mode*, welche mit Hilfe des Python-moduls *pexpect* und des Linux-Programms *gatttool* aus dem BlueZ-Stack die entsprechende Änderung des Working-Modes an den ESP32-Boards per Befehl in Abbildung 57 – Zeile 11 durchführt. Das Modul *pexpect* ermöglicht hierbei über Python eine gesteuerte Abfolge von Befehlen und Zeileneingaben in einer Linux Shell durchzuführen. Dies ist nötig, da *gatttool* interaktiv zu bedienen ist und somit diese Aufgabe nicht über ein Bash-Skript erfolgen kann. Bei dieser Methodik handelt es sich um einen Workaround, da dies mit puren Python Mitteln zum gegebenen Zeitpunkt nicht fehlerfrei durchgeführt werden konnte.

```

55 class MyCallbacks: public BLECharacteristicCallbacks {
56     void onWrite(BLECharacteristic *pCharacteristic) {
57         Serial.println("I am Here");
58         std::string rxValue = pCharacteristic->getValue();
59         Serial.println("something has been written");
60
61         if(rxValue.length() == 4) {
62
63             std::string code;
64             code.append(1,rxValue[0]);
65             code.append(1,rxValue[1]);
66             code.append(1,rxValue[2]);
67
68             if (code == "WRT") {
69
70                 Serial.println("AUTH SUCCESSFULL");
71                 working_mode = rxValue[3];
72
73             }
74         }
75     }
76

```

Abbildung 58 - Implementierung des GATT Services für working_mode überschreibung

Zur einfachen Authentifizierung mit dem ESP32-Board werden die Bytes *0x57,0x52,0x54*, welche in ASCII „*WRT*“ ergeben und für WRITE stehen sollen, versendet mit anschließendem Byte für den Working Mode. Abbildung 58 zeigt hierbei auszugsweise die Implementierung dieses Vorgangs im Sketch der ESP32-Boards.

2.4 People-Counting

Als zusätzliches Instrument der Raumbelegungsanalyse dient ein People Counter System, welches anhand von Bildaufnahmen menschliche Strukturen erkennen und zählen kann. Die erarbeitete Lösung verwendet hierbei ein selbstentwickeltes System auf Basis einer Client-Server Architektur, dessen Funktionsweise in den folgenden Abschnitten näher beschrieben wird.

2.4.1 Erstellen von Bildaufnahmen

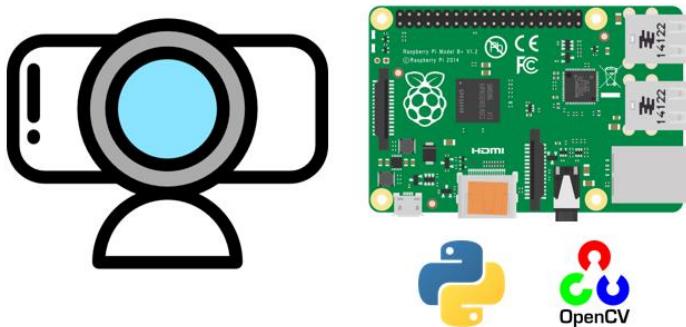


Abbildung 59 – Skizze People Counter Edge Device

Zum Erstellen von Bildaufnahmen kommt innerhalb der erarbeiteten Lösung ein Raspberry Pi 3B+ mit angeschlossener USB-Kamera zum Einsatz.

```
1 #General Config of the People-counter-behavior
2 general_config:
3     start_hour: 0
4     end_hour: 24
5     use_camera_id : 0
6     #possible send_modes: MQTT, INFLUX
7     send_mode : MQTT
8     wait_seconds_short: 60
9     wait_seconds_long: 300
10    sleep_seconds: 6000
11    sw_picture : false
12
13 #Config for OpenPose Endpoint
14 openpose_config:
15     server : 192.168.98.128
16     port : 8443
17     protocol: https
18     #Possible working_modes : JSON, FILE
19     working_mode : JSON
```

Abbildung 60 - Ausschnitt aus People Counter Edge Konfiguration

Wie auch alle anderen Python-Skripte der erarbeiteten Lösung, ist auch dieses Glied des Systems über eine Konfigurationsdatei konfigurierbar. Abbildung 60 zeigt einige Parameter, welche innerhalb der Konfigurationsdatei gesetzt werden können. Unter anderem kann festgelegt werden in welchem Zeitraum Aufnahmen erstellt werden sollen, ob ein Bild vor der Auswertung in Schwarz/Weiß umgewandelt wird oder ob das ermittelte Ergebnis per MQTT oder INFLUX-API übertragen werden soll.

Der Zeitraum soll dazu dienen, dass z.B. keine Aufnahmen bei Nacht oder außerhalb der Geschäftszeiten erstellt und ausgewertet werden sollen, denn dies spart Systemressourcen. Um das People Counting zu beschleunigen empfiehlt es sich, dass Aufnahmen vor der Auswertung in ein Schwarz/Weiß Bild konvertiert werden. Jedoch kann dies bei schwacher Raumbeleuchtung dazu führen, dass die Personenerkennung darunter leidet, weswegen die Konvertierung optional eingestellt werden kann. Damit für das People Counting nicht zwingend ein MQTT-Broker benötigt wird, ist es ebenfalls konfigurierbar, dass die ermittelten Ergebnisse direkt in eine Influx-Datenbank geschrieben werden können. Zudem kann die Art der Übertragung für die Aufnahmen über den Parameter *working_mode* definiert werden.

Abbildung 61 - Aufnahme per Kamera und Konvertierung in Base64

Abbildung 61 zeigt die Implementierung der Funktion *take-picture* und *image2json*. Wie der Name bereits vermuten lässt, ist die Funktion *take-picture* für die Aufnahme von Bildern über die am Raspberry Pi 3B+ angeschlossene USB-Kamera mittels OpenCV verantwortlich. In den Zeilen 108-114 werden zunächst plattformspezifische Treibereinstellungen [67] für die USB-Kamera vorgenommen, um den Aufnahmeprozess zu optimieren. Anschließend wird in Zeile 117 ein Frame von der USB-Kamera aufgenommen und als Rückgabewert gespeichert. Bildaufnahmen werden in OpenCV als Numpy-Arrays gespeichert und können auf diesem Wege weiterverarbeitet werden. Je nach Parametern der Konfigurationsdatei wird die Aufnahme anschließend in ein Schwarz/Weiß Bild konvertiert und anschließend als Bilddatei temporär im PNG-Format gespeichert oder in ein JSON-Objekt konvertiert. Die Umwandlung in ein JSON-Objekt ist über die Funktion *image2json* implementiert, welche das Numpy-Array der Bildaufnahme in ein Base64-ASCII [68] String umwandelt und zusammen mit den Dateinamen als JSON-Objekt, wie in *Abbildung 61* beispielhaft dargestellt, zurückgibt.

```

182     #get current unix-timestamp
183     timestamp = int(time.time())
184
185     #set a filename for the temporary picture
186     filename = "PeopleCount_"+str(timestamp)+".png"
187
188     #Take Picture base on openPose Working Mode
189     if OPENPOSE_WORKING_MODE.upper() == "FILE":
190
191         pic = take_picture(KAMERA_ID,filename,mode="FILE")
192         #send to openPose-Endpoint for counting
193         print("{} | WAITING FOR RESULT...".format(get_log_time()))
194         result = get_people_file(OPENPOSE_SERVER,filename)
195         #delete old Image after counting
196         os.remove(filename)
197
198     if OPENPOSE_WORKING_MODE.upper() == "JSON":
199
200         pic = take_picture(KAMERA_ID,filename,mode="JSON")
201         print("{} | WAITING FOR RESULT...".format(get_log_time()))
202         if pic != False:
203             result = get_people_json(OPENPOSE_SERVER,pic)
204         else:
205             result = -1
206
207         print("{} | COUNTED PEOPLE: {}".format(get_log_time(),result))
208

```

Abbildung 62 - Übertragung an OpenPose Server

Je nach konfiguriertem *working_mode* werden anschließend die Bildaufnahmen an einen OpenPose Server zum People Counting übertragen. Da dies je nach Rechenleistung des OpenPose Servers und der Menge der Bildinformationen einige Sekunden bis hin zu mehreren Minuten in Anspruch nehmen kann, wird vor diesem Vorgang der aktuelle Zeitpunkt per Unix-Timestamp (*Abbildung 60 – Zeile 183*) erfasst. Dieser dient nach dem People Counting als Zeitpunkt des Messwertes und wird zusätzlich per MQTT oder Influx-API übertragen.

Um das Ergebnis des People Counting zu verbessern hat sich gezeigt, dass die Kamera mindestens in einer Höhe von zwei Metern im Raum platziert werden sollte, sodass die Aufnahme den Raum und die darin befindlichen Personen von oben erfasst. Dadurch wird die Möglichkeit, dass sich Personen auf einer Aufnahme gegenseitig verdecken (*siehe Abbildung 61*), weitestgehend minimiert und das People Counting läuft zuverlässiger.

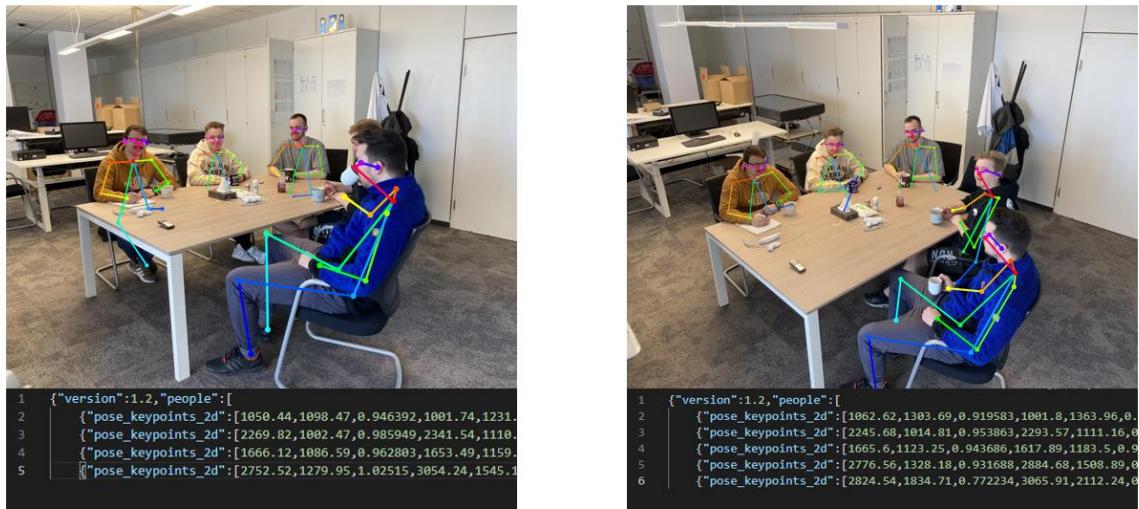


Abbildung 63- Ergebnisse von OpenPose bei unterschiedlicher Kamerahöhe (links 1.5m, rechts 2.3m)

2.4.2 Verarbeiten von Bildaufnahmen mit OpenPose

```
1 #General Config of the Serverbehavior
2 flask_config:
3     host : '0.0.0.0'
4     port : 8443
5     ssl_context : 'adhoc'
6     upload_folder : '/etc/people-counter-server/docker_temp'
7
8 #OpenPose configuration
9 openpose_config:
10    dockermode : false
11    docker_container : 'openpose'
12    #disable to display result with value 0
13    op_param_display : 0
14    #OpenPose Model to use: COCO / BODY_25
15    op_param_model_pose : 'COCO'
16    #net resolution must by a multiple of 16 (can increase accuracy)
17    op_param_net_resolution : '-1x256'
18    #disable openpose logging with value 255
19    op_param_logging_level : 255
20    #special for OpenPose behavior (optional)
21    #reference: https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/include/openpose/flags.hpp
22    op_param_scale_number : 1.0
23    op_param_scale_gap : 0.25
24    op_param_model_folder : '/home/alex/openpose/models'
25    op_param_python_folder : '/usr/local/python'
26    op_param_use_faces : false
27    op_param_use_hands : false
```

Abbildung 64 - OpenPose Server Konfiguration

Wie bereits beschrieben wird das People Counting über eine REST-API bereitgestellt, welche auf einem Ubuntu Server 20.04 LTS VM per FLASK implementiert ist. Die REST-API nimmt die übertragenen Bildinformationen per POST-Request entgegen und verarbeitet diese anschließend mit OpenPose. Da innerhalb der Ubuntu Server VM keine GPUs (Graphics Processing Unit) zur Verfügung stehen, wird OpenPose im CPU-Modus kompiliert und betrieben, was sich spürbar auf die Performance von OpenPose auswirken kann. Daher wird OpenPose mit entsprechenden Parametern bzw. Flags ausgeführt, welche einen guten Kompromiss aus Geschwindigkeit und möglichst hoher Genauigkeit beim People Counting gewährleisten. So empfehlen die Entwickler von OpenPose innerhalb ihres GitHub-Repositories für eine höhere Akkuranz das *BODY_25-Modell* zu benutzen, weisen aber darauf hin, dass das *COCO-Modell*, welches in Abbildung 62 – Zeile 15 konfiguriert ist, sich für die Nutzung im CPU-Modus besser eignet [69]. Mittels des *net_resolution - Parameters* (Abbildung 62 – Zeile 17) kann zudem noch die Erkennungsgeschwindigkeit weiter optimiert werden. Dabei muss der angegebene Wert einem Vielfachen von 16 entsprechen. Je höher der Wert, desto genauer „kann“ die Akkuranz werden, jedoch zu Lasten von benötigten Ressourcen. Im Standard ist die *net_resolution* auf -1x368 eingestellt. Jedoch zeigt sich, dass diese Einstellung im CPU-Modus signifikant mehr Ressourcen und Zeit benötigt als z.B. die gewählte Einstellung -1x256 und zudem keine erkennbar genaueren Ergebnisse liefert. Die restlichen Einstellungen können optional genutzt werden und entsprechen bereits den Standardwerten von OpenPose. Sie können für zukünftige Entwicklungen genutzt werden um z.B. explizit Gesichtspunkte wie Augen, Nase, Mund, etc. oder Hände zu erkennen. Diese werden aber im Rahmen der erarbeiteten Lösung nicht in Betracht gezogen.

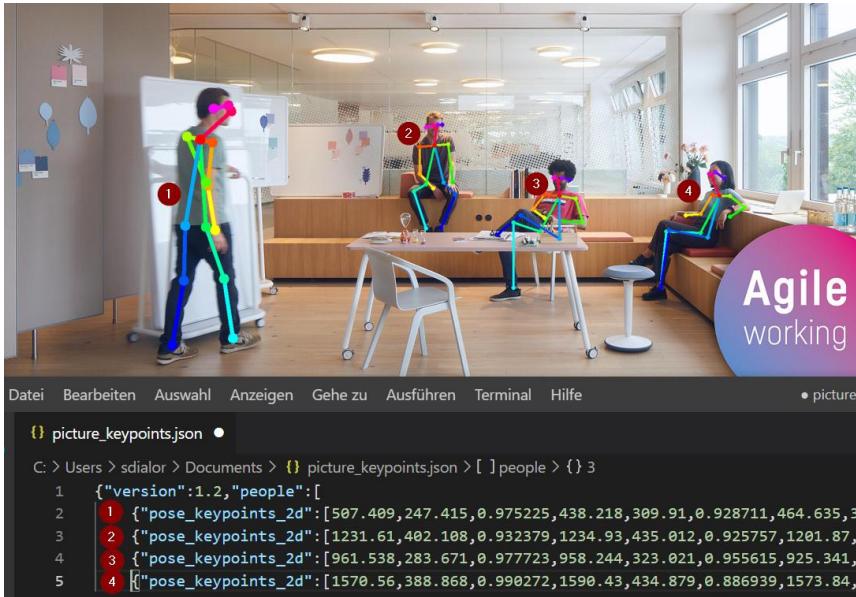


Abbildung 65 - OpenPose Ergebnisbild in erstellter Keypoints.json-Datei

Abbildung 65 zeigt demonstrativ ein von OpenPose mit dem COCO-Modell analysiertes Bild inklusive der erstellten *Pose_Keypoints* in einer JSON-Datei. Die erkannten menschlichen Personen werden dabei in einer Liste aus *pose_keypoints* definiert. Zum Zählen der Personen auf einem Bild kann somit die Anzahl des Keys *pose_keypoints_2d* verwendet und dem Aufrufer der REST-API als Response zurückgegeben werden. Im Beispiel von Abbildung 65 wäre dies der Wert 4.

Die erarbeitete Lösung bietet für die Verarbeitung zwei verschiedene Modi, welche in den folgenden Abschnitten kurz erläutert werden.

2.4.2.1 People Counting via Docker-Container

```

52 #function for peoplecountig inside a docker container
53 def count_people_docker(folder,jsonfilename):
54
55     print('{}: Start - Imageprocessing {}'.format(get_current_time(),str(folder)))
56     try:
57         dockerstring = 'docker run --rm -v {}:/data {} -display 0 \
58             -image_dir /data -write_images /data -write_json /data -model_pose {} \
59             -net_resolution {} -scale_number {} -scale_gap {} \
60             >> /dev/null'.format(folder,DOCKER_CONTAINER,OPENPOSE_MODEL,OPENPOSE_NET_RESOLUTION,OPENPOSE_SCALE_NUMBER,OPENPOSE_SCALE_GAP)
61         os.system(dockerstring)
62         #jsonfile = open('C:\\temp\\images\\'+jsonfilename)
63         jsonfile = open(os.path.join(folder,jsonfilename))
64         openpose_result = json.load(jsonfile)
65         jsonfile.close()
66         shutil.rmtree(folder)
67         print('{}: Done - Imageprocessing {}'.format(get_current_time(),str(folder)))
68         return len(openpose_result['people'])
69     except:
70         print('{}: Error - Imageprocessing {}'.format(get_current_time(),str(folder)))
71         if os.path.exists(folder):
72             shutil.rmtree(folder)
73         return -1

```

Abbildung 66 - People Counting mit Docker (Methode: count_people_docker)

In diesem Modus wird das People Counting mit OpenPose innerhalb eines Docker Container realisiert. Das verwendete Docker Image stammt vom Entwickler Sean Cook und ist auf GitHub zu finden [70]. Dieser Modus hat den Vorteil, dass er sehr schnell zu implementieren und auch durch die Möglichkeiten die Docker in Themen wie z.B. Lastenverteilung bietet, sehr skalierbar ist. Der große Nachteil ergibt sich jedoch dadurch, dass für das People Counting zwingend Bildaufnahmen in einem gängigen Bildformat wie z.B. JPG oder PNG auf dem OpenPose-Server vorliegen müssen.

```
root@degebachelor01:~# docker run --rm -v /home/alex:/data openpose -display 0 \
> -image_dir /data -write_images /data -write_json /data -\
> model_pose COCO -net_resolution -lx256
```

Abbildung 67 - People Counting Docker Befehl

Abbildung 67 zeigt beispielhaft die Verwendung von OpenPose mit Docker. Das Prinzip des People Counting beruht darauf, dass mittels des Parameters „-v /home/alex:/data“ der Ordner /home/alex des Docker Hosts zur Laufzeit des Docker Containers als „/data“ in den Container als Volume gemountet wird. Anschließend wird OpenPose innerhalb des Docker Containers mit den in Abbildung 67 zu sehenden Parametern ausgeführt. Unter anderem wird mit „-image_dir“ auf das gemountete Volume verwiesen, in welchem sich ein oder mehrere Bilder befinden können. Für das People Counting interessiert uns primär der Parameter „-write_json /data“, welcher dafür sorgt, dass die gefundenen Personen als Keypoints innerhalb einer JSON-Datei, wie in Abbildung 68 zu sehen, aufgelistet werden.

```
1 ↵ {"version":1.2,"people": [
2   |  {"pose_keypoints_2d":[1062.62,1303.69,0.919583,1001.8,1363.96,0.9
3   |  |  {"pose_keypoints_2d":[2245.68,1014.81,0.953863,2293.57,1111.16,0.
4   |  |  {"pose_keypoints_2d":[1665.6,1123.25,0.943686,1617.89,1183.5,0.92
5   |  |  {"pose_keypoints_2d":[2776.56,1328.18,0.931688,2884.68,1508.89,0.
6   |  |  {"pose_keypoints_2d":[2824.54,1834.71,0.772234,3065.91,2112.24,0.
```

Abbildung 68 - OpenPose Keypoints in JSON

Nachdem der OpenPose Befehl abgeschlossen ist, wird der Docker Container wieder gelöscht. Dies ist nötig, da der OpenPose Server als zentrale People Counter Instanz konzipiert ist und theoretisch von mehreren People Counter Edge Devices genutzt werden kann. Um nicht jedem People Counter Edge Device einen festen Container zuordnen zu müssen, erstellt daher jedes People Counting Request einen eigenen temporären Container mit eigener temporärer Verzeichnisstruktur auf dem OpenPose Server, welche nach Abarbeitung des People Countings aus Datenschutzgründen wieder gelöscht wird.

Die Anzahl der erkannten Personen ergibt sich nun aus der Anzahl der Objekte in der Liste von *people*, welche in Abbildung 66 – Zeile 2-6 zu sehen sind. In Abbildung 66 – Zeile 64 wird hierfür zunächst aus der von OpenPose erstellten JSON-Datei ein Python-Dictionary erstellt und anschließend in Zeile 68 mit *return len(openpose_result['people'])* die Anzahl der darin befindlichen Objekte zurückgegeben.

Im Falle, dass bei der Personenermittlung ein Fehler auftreten sollte, wird der Wert -1 zurückgegeben (Abbildung 66 – Zeile 73).

2.4.2.2 People Counting via OpenPose Python Wrapper

```

72 #function for peoplecounting directly with the OpenPose-API
73 def count_people_openpose_native(image):
74     try:
75         sys.path.append(OPENPOSE_PYTHON)
76         from openpose import pyopenpose as op
77
78         params = dict()
79         params["model_folder"] = OPENPOSE_MODEL_FOLDER
80         params["face"] = False
81         params["hand"] = False
82         params["display"] = 0
83         params["model_pose"] = OPENPOSE_MODEL
84         params["net_resolution"] = OPENPOSE_NET_RESOLUTION
85         params["scale_number"] = OPENPOSE_SCALE_NUMBER
86         params["scale_gap"] = OPENPOSE_SCALE_GAP
87         params["logging_level"] = OPENPOSE_LOGGING_LEVEL
88
89         opWrapper = op.WrapperPython()
90         opWrapper.configure(params)
91         opWrapper.start()
92
93         datum = op.Datum()
94         datum.cvInputData = image
95
96         opWrapper.emplaceAndPop(op.VectorDatum([datum]))
97
98         if type(datum.poseKeypoints) != type(None):
99             return (len(datum.poseKeypoints))
100        else:
101            return 0
102    except:
103        return -1

```

Abbildung 69- People Counting nativ via Python (Methode: `count_people_openpose_native`)

In diesem Modus wird das People Counting mit OpenPose nativ aus Python heraus auf dem OpenPose-Server ausgeführt. Abbildung 69 zeigt die Implementierung von OpenPose in Python. Um OpenPose aus Python heraus nutzen zu können, müssen bei der Kompilierung, die in der Abbildung 70 grün markierten Parameter für die Implementierung des Python-Wrappers von OpenPose mit angegeben werden.

```

12 sudo cmake -DGPU_MODE:String=CPU_ONLY -DDOWNLOAD_BODY_MPI_MODEL:Bool=ON \
13 -DDOWNLOAD_BODY_COCO_MODEL:Bool=ON -DDOWNLOAD_FACE_MODEL:Bool=ON -DDOWNLOAD_HAND_MODEL:Bool=ON \
14 -DUSE_PYTHON_INCLUDE_DIR=ON -DBUILD_PYTHON=ON -DPYTHON_EXECUTABLE=/usr/bin/python3 ..

```

Abbildung 70 - Kompilieren von OpenPose mit Python Wrapper

Der Vorteil gegenüber der Docker Variante ergibt sich dadurch, dass OpenPose nun direkt aus Python heraus mit Bildinformationen in Form eines Numpy-Arrays, wie sie auch bei OpenCV zum Einsatz kommen, umgehen und die ermittelten KeyPoints direkt als Python-Dictionary speichern kann. Dieses Python-Dictionary beinhaltet die gleichen Informationen wie die erzeugte JSON-Datei der Docker Variante und kann auf gleiche Art und Weise dazu genutzt werden um die Anzahl erkannter Personen zu ermittelt und zurückzugeben. Dadurch, dass nun keine Bildaufnahmen in Form einer JPG oder PNG Datei vorliegen müssen, ist diese Variante zudem noch Datenschutzkonform [49], da die zu verarbeitenden Aufnahmen sich nur im Arbeitsspeicher des OpenPose-Servers befinden. Gegenüber der Docker Variante könnte hier OpenPose mit GPU-Support betrieben werden, was die Verarbeitungsgeschwindigkeit und die Akkuranz deutlich beschleunigen würde.

2.5 Speichern und Visualisieren der gewonnenen Sensordaten

Nachdem von den Edge Devices alle Sensordaten erfasst wurden, leiten diese die gewonnenen Daten an den MQTT – InfluxDB – Broker weiter. Dies kann je nach Konfiguration der Edge Devices per MQTT oder InfluxDB-API erfolgen. Beim MQTT – InfluxDB – Broker handelt es sich in der erarbeiteten Lösung um einen Ubuntu Server 20.04 LTS VM, welcher einen Mosquito MQTT-Broker und einen InfluxDB-Server als Service betreibt.

Abbildung 71 - Main Loop von mqtt-to-influx-broker.py (links) / mqtt_config Parameter für MQTT-to-Influx-Broker (rechts)

Für die Datenübertragung der Sensordaten vom MQTT-Broker in eine InfluxDB ist das Python Skript *mqtt-to-influx-broker.py* zuständig. Abbildung 71 (links) zeigt hierbei den *Main Loop* in verkürzter Form. In diesem wird zunächst ein MQTT-Client Objekt erzeugt (Zeile 178), welches anschließend eine Verbindung zum MQTT-Broker aufbaut (Zeile 186), alle MQTT-Tops aus der Konfigurationsdatei (Abbildung 69 – rechts - Zeile 19 bis 38) subscribed und auf Messages des MQTT-Brokers reagiert (Abbildung 69 – links - Zeile 207 - 208). Zu Beachten ist, dass innerhalb der Konfiguration in den MQTT-Tops auch Wildcards in Form von „+“ und „#“ verwendet werden dürfen.

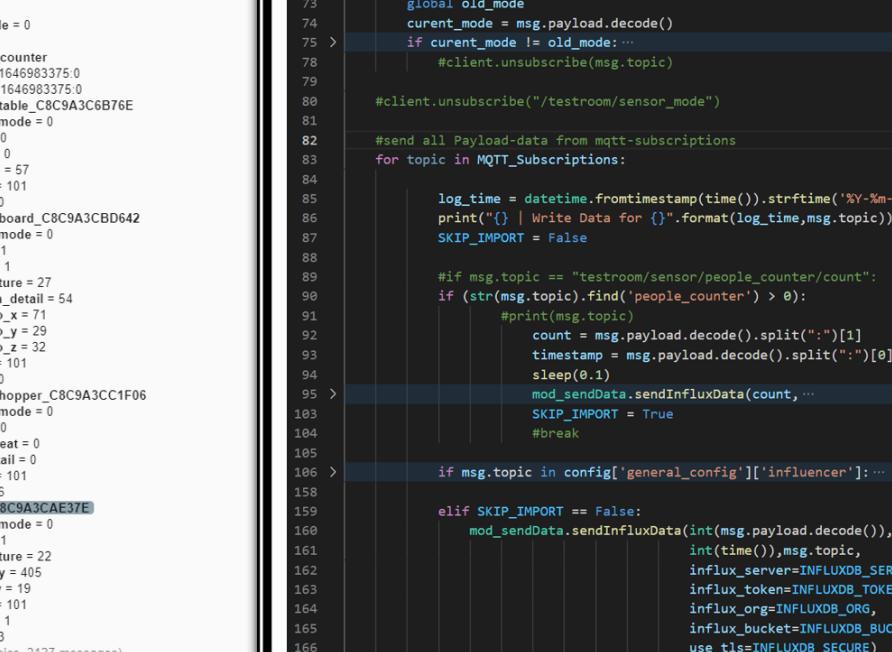


Abbildung 72 - Beispiel Topic Levels (Quelle: www.hivemq.com)

Das „+“ Symbol dient als Single-Level Wildcard, welches zwischen zwei Topic Level verwendet werden kann. So ist es z.B. möglich über das Topic „*testroom/sensor/+rss*“ den RSSI (Received Signal Strength Indicator) jedes Sensors aus dem Testroom zu subscriben.

Das „#“ Symbol hingegen dient als Multi-Level Wildcard, welches als letztes Glied eines Topic Levels benutzt werden kann, um alle darauffolgenden Topic Level zu subscriben. So wäre es z.B. über das Topic „*testroom/sensor/#*“ möglich alle Topics der Sensoren im testroom zu subscriben.

Dies vereinfacht und minimiert den Umfang der MQTT Konfiguration enorm [71].



The screenshot shows the MQTT Explorer interface with a device tree on the left and a code editor on the right.

Device Tree (Left):

- 192.168.1.177
 - testroom
 - sensor_mode = 0
 - sensor
 - people_counter
 - count = 1646983375:0
 - usage = 1646983375:0
 - se_lab_table_C8C9A3C6B76E
 - sensor_mode = 0
 - usage = 0
 - trigger = 0
 - distance = 57
 - battery = 101
 - rssi = .70
 - se_lab_board_C8C9A3CBD642
 - sensor_mode = 0
 - usage = 1
 - trigger = 1
 - temperature = 27
 - vibration_detail = 54
 - diff_gyro_x = 71
 - diff_gyro_y = 29
 - diff_gyro_z = 32
 - battery = 101
 - rssi = -80
 - se_lab_hopper_C8C9A3CC1F06
 - sensor_mode = 0
 - usage = 0
 - usage_seat = 0
 - seat_detail = 0
 - battery = 101
 - rssi = -66
 - room_C8C9A3CAE37E
 - sensor_mode = 0
 - usage = 1
 - temperature = 22
 - airquality = 405
 - humidity = 19
 - battery = 101
 - trigger = 1
 - rssi = -73
 - \$SYS (45 topics, 2127 messages)
 - cisco_room
 - sensor
 - people_counter
 - count = 1646983411:0
 - usage = 1646983411:0

Abbildung 73 – MQTT-Topics (links) | Implementierung „on message“-Callback verkürzt (rechts)

Sobald ein Edge-Device einen neuen Wert auf den MQTT-Broker published, sendet dieser eine Message an alle Subscriber eines Topics raus. Das erzeugte MQTT-Client Objekt kann auf diese „on_message“ – Callbacks reagieren und die erhaltenen Payloads (Daten) innerhalb des Python-Skripts verarbeiten.

Abbildung 73 zeigt eine verkürzte Ansicht der Implementierung des „on_message“ – Callbacks. Das Python-Skript ist so konzipiert, dass es MQTT-Topic spezifisch die Payload Daten für den Import in die zuvor angegebene InfluxDB aufbereitet und importiert. Zudem erfolgt hier auch der Wechsel der working_modes, welcher durch die Sensoren mit den Rollen influencer und swichtmaster vorgegeben wird.

```
▼ sensor
  ▼ people_counter
    count = 1646983615:0
    usage = 1646983615:0
```

Abbildung 74 - Peoplecounter MQTT-Payload

Eine Besonderheit der Payload-Data stellt z.B. der People Counter dar. Abbildung 74 zeigt, dass dieser Payload keinem Standard Integer oder Float Wert entspricht, sondern als String in der Form „*UnixTimeStamp:Value*“ an den MQTT-Broker gepublished wird. Dies hat den Grund, dass das Ergebnis des People Countings teilweise erst einige Minuten später vom OpenPose-Server zurückgegeben werden kann. Daher wird der Unix Timestamp bei Bildaufnahme des People Counters als Zeitpunkt für die Messung zusätzlich zum Ergebnis hinzugefügt und muss im „*on_message*“ – Callback auch separat betrachtet und ausgewertet werden (Abbildung 73 - rechts – Zeile 90-104).

Anschließend werden die gewonnenen Daten aus dem MQTT-Payload an die Methode `sendInfluxData` aus dem Modul `mod_sendData` zum Speichern in InfluxDB übergeben.

```

10  # You can generate an API token from the "API Tokens Tab" in the InfluxDB WebUI
11 def sendInfluxData(data,timestamp,
12                     mqtt_topic=None,
13                     influx_server="192.168.178.36",
14                     influx_port=8086,
15                     influx_bucket="sensor_data",
16                     influx_org="sedus",
17                     influx_token="cWFmcPImccIW0cnGn9ewKmYQxxXiNSJsPFHDMapNhGFqQfTrN7cIw9HVUOYus1-ujof1jtWSEdx0o9NRl7yHhw==",
18                     use_tls = False
19                     ):
20
21     #set server URL based on encryption
22     if (use_tls == True):
23         server = "https://{}:{}".format(influx_server,influx_port)
24     else:
25         server = "http://{}:{}".format(influx_server,influx_port)
26
27     with InfluxDBClient(url=server, token=influx_token, org=influx_org,ssl=True, verify_ssl=False) as client:
28
29         #setup InfluxDB client
30         write_api = client.write_api(write_options=SYNCHRONOUS)
31
32         #if now MQTT-Topic is given write simply the data
33         if mqtt_topic == None:
34
35             else:
36                 influx_points=mqtt_topic.split('/')
37
38                 #check if the MQTT-Topics are in the wished format
39                 if (len(influx_points) <4):
40                     print("Invalid mqtt-format!")
41                     return "err"
42
43                 #Set the Datapoints for InfluxDB
44                 point = Point(influx_points[0]) \
45                     .tag(influx_points[1], influx_points[2]) \
46                     .field(influx_points[3], float(data)) \
47                     .time(datetime.utcnow().fromtimestamp(int(timestamp)), WritePrecision.NS)
48
49
50                 #Write Datapoint to InfluxDB
51                 write_api.write(influx_bucket, influx_org, point)
52
53
54
55

```

Abbildung 75 - Implementierung `sendInfluxData`-Methode in `mod_SendData.py`

Abbildung 75 zeigt verkürzt die Implementierung der `sendInfluxData` Methode aus dem Modul `mod_SendData`. Diese Methode sorgt dafür, dass Daten, welche aus dem MQTT-Payload gewonnenen werden, auch passend formatiert in eine InfluxDB per InfluxDB-API übertragen werden. In den Zeilen 11 – 18 ist zu sehen, dass dafür der Methode eine Reihe von Parametern mitgeben werden kann. Die zu sehenden Parameter aus Zeilen 13-18 aus der Abbildung 75 sind zu Entwicklungszwecken hart kodiert, werden jedoch für den Methodenaufruf aus der `mqtt-to-influx-broker.py` aus der Konfigurationsdatei des Dienstes herausgelesen und explizit für die passende Systemumgebung mitgegeben.

Somit repräsentieren die Parameter `data`, `timestamp` und `mqtt_topic` (Zeilen 11-12) die relevanten Daten der erarbeiteten Lösung. Der Parameter `data` beinhaltet dabei den ermittelten Wert eines Sensors. Beim Parameter `timestamp` wird ein Zeitpunkt im Unix Timestamp-Format mitgegeben, welcher den Zeitpunkt einer Messwertermittlung definiert. Um die ermittelten Sensorwerte auch einem Sensor zuordnen zu können wird noch zusätzlich das dazugehörige MQTT-Topic an den Parameter `mqtt_topic` mit übergeben.

Die Datenablage in InfluxDB ist dabei wie folgt organisiert:

Jede Instanz in InfluxDB wird als *Organization* bezeichnet und kann mehrere *Buckets* beinhalten. Innerhalb dieser *Buckets* können mehrere *Series* als *Datapoints* gespeichert werden. Ein *Datapoint* beinhaltet ein *Measurement*, welches die Daten eines *Datapoints* beschreibt. Zusätzlich zum *Measurement* befinden sich im *Datapoint* noch *Tags* welche die Daten eines *Measurements* noch detaillierter beschreiben können. Ein *Field* repräsentiert letzten Endes, um was es sich namentlich bei einem Wert innerhalb eines *Datapoints* genau handelt und beinhaltet innerhalb der erarbeiteten Lösung einen Float-Wert.

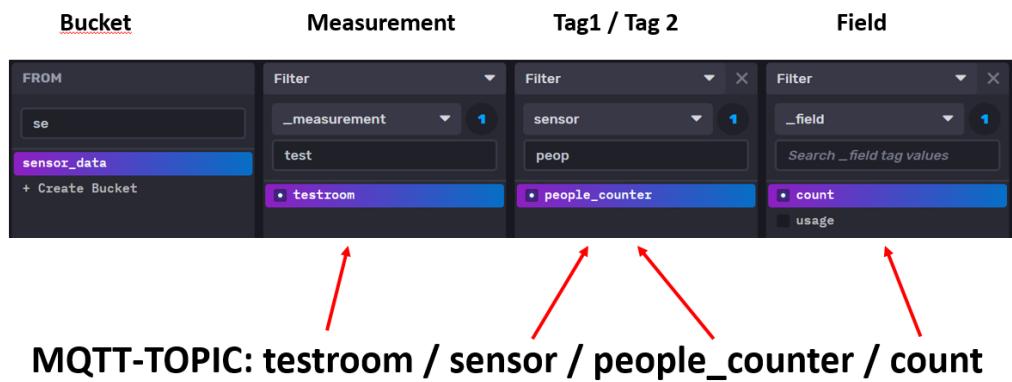


Abbildung 76 - Schaubild InfluxDB Schema und MQTT-Topic

Damit die gewonnenen Sensordaten in InfluxDB geschrieben werden können, haben die gewünschten MQTT-Topics innerhalb der erarbeiteten Lösung immer vier Topic Level, wie im Abbildung 76 skizziert. Sollte ein MQTT-Topic nicht aus vier Topic Leveln bestehen, so wird dieser Datensatz nicht in InfluxDB geschrieben und die Messung geht verloren (Abbildung 75 – Zeile 41 – 46).

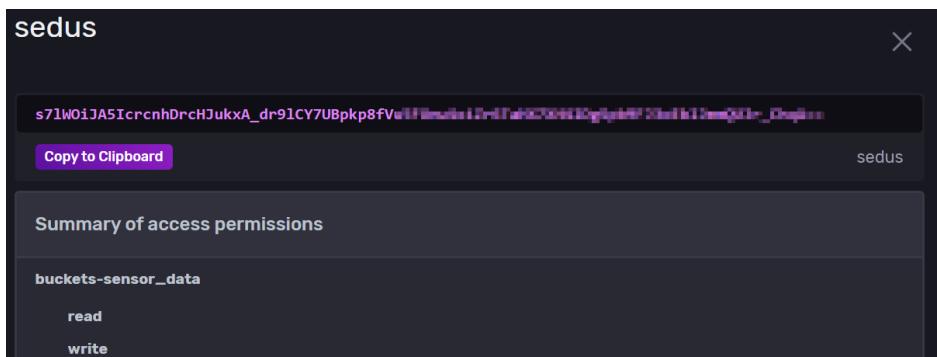


Abbildung 77 - InfluxDB-API Token in InfluxDB Web-Frontend

Damit ein Datensatz per InfluxDB-API in InfluxDB geschrieben werden kann, wird ein API-Token benötigt. Dieser kann unter anderem innerhalb des InfluxDB-Webfrontend erstellt werden. Abbildung 77 zeigt hierbei einen erstellten API-Token welcher Lese- und Schreibzugriff auf den Bucket *sensor_data* innerhalb der Organisation *sedus* hat.

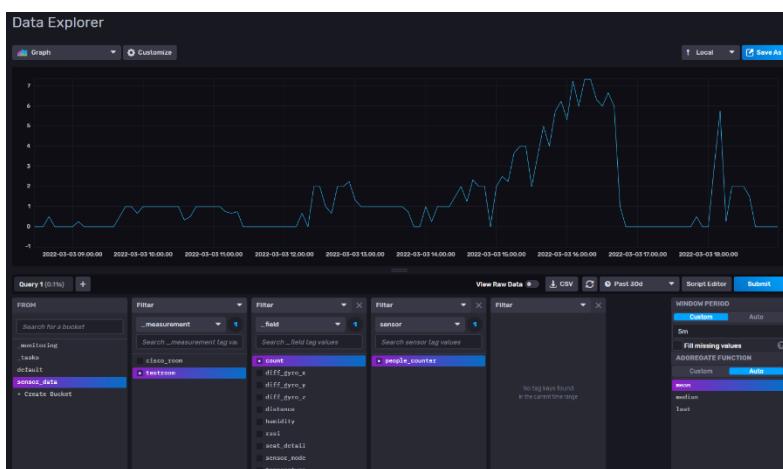


Abbildung 78 – People Counter Graph im InfluxDB Data Explorer

Anschließend können die gewonnenen Daten im InfluxDB-Webfrontend unter Data Explorer erkundet und ausgewertet werden.



Abbildung 79 - Dashboard in Grafana

Alternativ kann die InfluxDB natürlich auch in eine Datenvisualisierungslösung wie *Grafana* [72] eingebunden werden, um wie in Abbildung 79 zu sehen, komplexere Dashboards zu erstellen.

2.6 Installation

```

1  #!/bin/bash
2
3  echo "Run this script with SUDO!!!!"
4
5  #Getting needed Resources
6  echo -n "Installing needed Resources..."
7  sudo apt-get update
8  sudo apt-get install python3-pip -y
9  sudo apt-get install python3-opencv -y
10 sudo apt-get install python3-pip -y
11 sudo pip install influxdb-client
12 sudo apt-get install python3-paho-mqtt -y
13 sudo apt-get install python3-yaml -y
14 echo "...DONE"
15
16 #Creating Folders
17 echo .
18 echo .
19 echo -n "Creating folders..."
20 sudo mkdir /etc/people-counter-client
21 sudo mkdir /etc/people-counter-client/ssl
22 sudo mkdir /opt/people-counter-client
23 sudo mkdir /var/log/people-counter-client
24 echo -n "...DONE"
25
26 #Copying Files
27 echo .
28 echo .
29 echo -n "Copying files ..."
30 sudo cp *.py /opt/people-counter-client/
31 sudo cp *.yaml /etc/people-counter-client/
32 sudo cp people-counter-client.service /lib/systemd/system/
33 echo "...DONE"
34
35 #Installing Service
36 echo .
37 echo .
38 echo -n "Creating takePicture-Service ..."
39 sudo chmod 644 /lib/systemd/system/people-counter-client.service
40 sudo systemctl daemon-reload
41 sudo systemctl enable people-counter-client.service
42 echo "...DONE"
43

```

```

1  [Unit]
2  Description=People Counter Client Service
3  After=multi-user.target
4
5  [Service]
6  Type=idle
7  ExecStart=/usr/bin/python -u /opt/people-counter-client/people-counter-client.py
8  StandardOutput=file:/var/log/people-counter-client/service.log
9  StandardError=file:/var/log/people-counter-client/error.log
10 Restart=on-failure
11
12 [Install]
13 WantedBy=multi-user.target

```

```

pi@raspberrypi: ~ systemctl status people-counter-client
● people-counter-client.service - People Counter Client Service
   Loaded: loaded (/lib/systemd/system/people-counter-client.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2022-03-14 05:59:57 CET; 2h 1min ago
     Main PID: 22049 (python)
        Tasks: 4 (limit: 1597)
           CPU: 36.509s
          CGroup: /system.slice/people-counter-client.service
                  └─22049 /usr/bin/python -u /opt/people-counter-client/people-counter-client.py

Mar 14 05:59:57 raspberrypi systemd[1]: Started People Counter Client Service.
pi@raspberrypi: ~
pi@raspberrypi: ~
pi@raspberrypi: ~ systemctl people-counter-client status
● people-counter-client.service - People Counter Client Service
   Loaded: loaded (/lib/systemd/system/people-counter-client.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2022-03-14 05:59:57 CET; 2h 1min ago
     Main PID: 22049 (python)
        Tasks: 4 (limit: 1597)
           CPU: 36.509s
          CGroup: /system.slice/people-counter-client.service
                  └─22049 /usr/bin/python -u /opt/people-counter-client/people-counter-client.py

Mar 14 05:59:57 raspberrypi systemd[1]: Started People Counter Client Service.
pi@raspberrypi: ~

```

Abbildung 80 - Installations-Skript und Service-File von people-counter-client

Um die Installation und Inbetriebnahme aller nötigen Komponenten der erarbeiteten Lösung zu vereinfachen, liegen in den Projektordnern der Server und Edge Devices jeweils ein *Bash-Installationsskript* mit zusätzlichem *Service-File* vor. In Abbildung 80 sind beispielhaft diese beiden Dateien für das BLE-MQTT-Gateway abgebildet. Die jeweiligen Installationsskripte ermöglichen, dass alle nötigen Bibliotheken, Module und Applikationen, welche zum Ausführen der Python-Skripte benötigt werden, automatisch aus dem Internet heruntergeladen und installiert werden.

Zudem wird das jeweilige Hauptskript als *Service-Daemon* [73] auf dem jeweiligen Linux System eingerichtet und gestartet. Dabei gilt zu beachten, dass für die Installation Root bzw. Sudo-Rechte vorhanden sein müssen.

Zur besseren Strukturierung und Organisation des erstellten Daemon befinden sich z.B. alle Konfigurationsdateien in einem Ordner unter */etc/*, wohingegen die Python-Skripte in einem Ordner unter */opt/* zu finden sind. Log-Dateien des Daemon werden in einem Ordner unter */var/log/* abgelegt. Dadurch, dass das jeweilige Hauptskript als Daemon eingerichtet ist, kann dieses unter anderem immer direkt beim Start des Linux-Systems ausgeführt werden. Zudem kann der jeweilige Daemon über den *systemctl* oder *service* Befehl beendet, neugestartet oder der aktuelle Status (*Abbildung 80*) eingesehen werden.

3 Verschlüsselung und Absicherung der Kommunikation

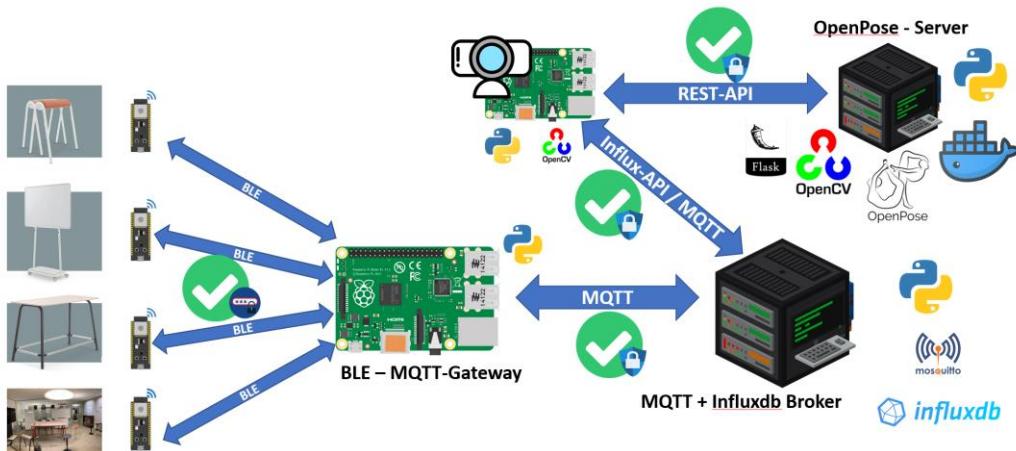


Abbildung 81- Verschlüsselte Kommunikation der erarbeiteten Lösung

Da die erarbeitete Lösung in ihrer Architektur so konzipiert ist, dass z.B. standortübergreifend Edge Devices eingesetzt werden können, ist eine Verschlüsselung des Datenverkehrs zwischen Edge und Server Devices wünschenswert. Aus diesem Grund können diese Kommunikationswege innerhalb der erarbeiteten Lösung optional verschlüsselt werden (*Abbildung 81*).

3.1 SSL-Verschlüsselung

```
root@degeinfluxdb:~# cat /etc/mosquitto/conf.d/ssl.conf
listener 8888
cafile /etc/mosquitto/ca_certificates/my_ca.crt
keyfile /etc/mosquitto/certs/my_server.key
certfile /etc/mosquitto/certs/my_server.crt
tls_version tlsv1.2
require_certificate false

root@degeinfluxdb:~# cat /etc/influxdb/config.toml
bolt-path = "/var/lib/influxdb/influxd.bolt"
engine-path = "/var/lib/influxdb/engine"
tls-cert = "/etc/ssl/influxdb-selfsigned.crt"
tls-key = "/etc/ssl/influxdb-selfsigned.key"
```

Abbildung 82 - SSL Konfigurationen v.l Mosquitto und InfluxDB

Für die Verschlüsselung des TCP-IP Datenverkehrs werden SSL-Zertifikate und die dazugehörigen privaten Schlüssel benötigt. Die Zertifikate können z.B. per *OpenSSL* selbst erstellt und signiert oder von einer anderen CA (Certified Authority) bereitgestellt werden [74]. Abbildung 82 zeigt hierbei die nötigen SSL-Konfigurationen für Services von Mosquitto und InfluxDB.

40	<code>mqtt_tls_enable: true</code>	53	<code>influxdb_port: 8086</code>
41	<code>mqtt_tls_port : 8888</code>	54	<code>influxdb_tls_enable: true</code>
42	<code>mqtt_tls_ca_cert_file: my_ca.crt</code>		

Abbildung 83 - SSL-Config für MQTT und INFLUXDB in Konfigurationsdatei von MQTT-InfluxDB-Broker

Sofern die Verschlüsselung dieser Kommunikationskanäle genutzt werden soll, muss dieses auch in den Konfigurationsdateien der Edge- und Server Devices eingetragen werden (*Abbildung 83*). Zusätzlich wird unter anderem bei der MQTT-Konfiguration das Root-Zertifikat der CA benötigt, da der *paho-mqtt-client* dieses für eine verschlüsselte Kommunikation benötigt. Die Verschlüsselung der Kommunikationskanäle sichert zudem, dass z.B. MQTT-Kennwörter oder API-Tokens nicht im Klartext übertragen werden und somit auch nicht abgefangen und mitgelesen werden können.

3.2 Absicherung MQTT

```
root@degeinfluxdb:~# echo influencer:SuperSecret >> mqtt_users
root@degeinfluxdb:~# sudo mosquitto_passwd -U mqtt_users
root@degeinfluxdb:~# cat mqtt_users
influencer:$6$XXa6G6KKdGvNIPzJ$Yu8/Err+D3cplquxlaGriel/i2Qm85tQtq0Toq4a/j/mlzarCSGpmfxFqw59WTREBsdI3t437nzCorWVAigYia==
root@degeinfluxdb:~# cat /etc/mosquitto/conf.d/users.conf
password_file /etc/mosquitto/mqtt_users
allow_anonymous false
```

Abbildung 84 - Mosquitto Benutzerberechtigung

Zur weiteren Absicherung bietet der Mosquitto MQTT-Broker eine Benutzerverwaltung mittels ACLs (Access Control List). Über diese ist es z.B. möglich Benutzern auf bestimmte MQTT-Topics Lese- und Schreibberechtigungen zu geben [75]. Auf diesem Wege kann unter anderem eine Sensorgenaue Berechtigungsstruktur aufgebaut werden, welche Edge Devices nur den Zugriff auf die für sie relevanten MQTT-Topics gewährt. *Abbildung 84* zeigt beispielhaft die nötigen Befehle und Konfigurationen zum Erstellen eines MQTT-Benutzers.

47	<code>mqtt_tls_user_name: influencer</code>
48	<code>mqtt_tls_user_pass: [REDACTED]</code>

Abbildung 85 - MQTT User Konfiguration für MQTT-Kommunikation

Sofern ein MQTT-User benutzt wird, kann dieser auch in den Konfigurationsdateien der Edge und Server Devices eingepflegt werden (*Abbildung 85*).

3.3 BLE Kennwörter

```
66      //Codeword for Writing the new Working_mode to the device
67      if (code == "WRT"){
68
69          Serial.println("AUTH SUCCESSFULL");
70          working_mode = rxValue[3];
71
72      }
73  }
```

Abbildung 86 - BLE Kennwort

Um das Ändern eines ESP32 – Working Modes nicht ganz ohne Sicherheitsmaßnahmen zu ermöglichen, ist es innerhalb der erarbeiteten Lösung nur mittels eines „Kennworts“ möglich, diesen zu ändern. Das Kennwort besteht hier beispielsweise aus den drei Character „WRT“, welche für das Wort Write stehen sollen, mit anschließend Working Mode, z.B. „WRT0“.

4 Ergebnisse



Abbildung 87- Grafana-Dashboard Raumnutzung

Durch die Implementierung der erarbeiteten Lösung als Proof-of-Concept ist eine Basis geschaffen worden, welche es ermöglicht, die Nutzung eines agilen Projektraums samt seiner Möbel, kabellos und sensorbasiert zu messen und die gewonnenen Daten über einen längeren Zeitraum hin zu bewerten. Auf Grund der konzipierten Systemarchitektur ist die erarbeitete Lösung zudem sehr flexibel erweiterbar und bietet durch Verschlüsselung einen akzeptablen Sicherheitsstandard bei der Übertragung von Sensordaten. Dadurch, dass OpenPose auf einem zentralen Server über eine REST-API genutzt werden kann ist zudem die Implementierung eines kostengünstigen People Counter Systems möglich, da die hierfür im Edge Device Bereich günstige Hardware eingesetzt werden kann. Die gewonnenen Erkenntnisse werden zudem in weitere Produktentwicklungen einfließen, um so Projektmöbel zukünftig noch smarter zu machen.

5 Aussichten

In ausgereifter Form kann die erarbeitete Lösung später unter anderem dafür genutzt werden, dass Sedus damit sensorgestützt seine Beratungs- und Planungsleistungen für Unternehmen erweitern und anbieten kann. Zudem wären dank der Nutzungserfassung zukünftig neue Geschäftsmodelle wie Pay-per-Use von agilen Projekträumen und Möbeln denkbar.

6 Literaturverzeichnis

- [1] D. Gebauer, Interviewee, *Personalabteilung Sedus*. [Interview]. 15 06 2021.
- [2] „Python 3.8.12 documentation,“ Python Software Foundation, 08 09 2021. [Online]. Available: <https://docs.python.org/3.8/>. [Zugriff am 05 01 2022].
- [3] „Visual Studio Code,“ Microsoft, 2022. [Online]. Available: <https://code.visualstudio.com/>. [Zugriff am 05 01 2022].
- [4] T. Kurek, „ubuntu.com blog,“ Canonical Ltd., 24 04 2020. [Online]. Available: <https://ubuntu.com/blog/ubuntu-server-20-04>. [Zugriff am 15 01 2022].
- [5] B. Lee, „nakivo.com,“ NAKIVO, Inc., 10 01 2017. [Online]. Available: <https://www.nakivo.com/blog/de/konfigurieren-eines-vmware-esxi-clusters/>. [Zugriff am 15 01 2022].
- [6] „VMware Workstation Pro,“ VMware Inc., 2022. [Online]. Available: <https://www.vmware.com/de/products/workstation-pro.html>. [Zugriff am 15 01 2022].
- [7] „Raspberry Pi 3 Model B+,“ Raspberry Pi Foundation, [Online]. Available: <https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/>. [Zugriff am 15 01 2022].
- [8] „Raspberry Pi OS,“ Raspberry Pi Ltd., 2022. [Online]. Available: <https://www.raspberrypi.com/documentation/computers/os.html>. [Zugriff am 15 01 2022].
- [9] „esp32-developmentboard,“ AZ-Delivery Vertriebs GmbH, [Online]. Available: <https://www.az-delivery.de/en/products/esp32-developmentboard>. [Zugriff am 05 01 2022].
- [10] „Arduino Download,“ Arduino Software, 2022. [Online]. Available: <https://www.arduino.cc/en/software>. [Zugriff am 05 01 2022].
- [11] „Was ist TLS (Transport Layer Security)?,“ Cloudflare, Inc., 2022. [Online]. Available: <https://www.cloudflare.com/de-de/learning/ssl/transport-layer-security-tls/>. [Zugriff am 15 01 2022].
- [12] mstahl, „Bluetooth LE (Low Energy): Das ist der Unterschied zu normalem Bluetooth,“ BurdaForward GmbH, 30 09 2019. [Online]. Available: https://praxistipps.chip.de/bluetooth-le-low-energy-das-ist-der-unterschied-zu-normalem-bluetooth_114239. [Zugriff am 13 01 2022].
- [13] „Mosquitto MQTT-Broker,“ Eclipse Foundation, 17 11 2021. [Online]. Available: <https://mosquitto.org/>. [Zugriff am 13 01 2022].
- [14] T. Joos, „Zeitreihendatenbanken für das Speichern von Messdaten,“ Vogel Communications Group, 15 10 2021. [Online]. Available: <https://www.storage-insider.de/zeitreihendatenbanken-fuer-das-speichern-von-messdaten-a-1034371/>. [Zugriff am 13 01 2022].
- [15] „InfluxData - Homepage,“ InfluxData Inc., 2022. [Online]. Available: <https://www.influxdata.com/>. [Zugriff am 13 01 2022].

- [16] „Github - OpenPose,“ CMU-Perceptual-Computing-Lab, 09 01 2022. [Online]. Available: <https://github.com/CMU-Perceptual-Computing-Lab/openpose>. [Zugriff am 13 01 2022].
- [17] „OpenCV,“ OpenCV team, 2022. [Online]. Available: <https://opencv.org/about/>. [Zugriff am 13 01 2022].
- [18] N. Iwamatsu, „<http://manpages.ubuntu.com/>,“ Canonical Ltd., 2019. [Online]. Available: <http://manpages.ubuntu.com/manpages/bionic/man1/gatttool.1.html>. [Zugriff am 13 01 2022].
- [19] T. H.-J. An, „BlueZ,“ BlueZ Project, 06 01 2022. [Online]. Available: <http://www.bluez.org/>. [Zugriff am 13 01 2022].
- [20] „Paho-Mqtt,“ Eclipse Foundation, 2021. [Online]. Available: <https://www.eclipse.org/paho/>. [Zugriff am 15 01 2022].
- [21] „InfluxDB_Client Repository,“ InfluxData, 10 01 2022. [Online]. Available: <https://github.com/influxdata/influxdb-client-python>. [Zugriff am 15 01 2022].
- [22] B. Ghimir, „geekflare.com - YAML,“ Geekflare, 11 01 2021. [Online]. Available: <https://geekflare.com/de/yaml-introduction/>. [Zugriff am 15 01 2022].
- [23] „Json Format,“ [Online]. Available: <https://www.json.org/json-de.html>. [Zugriff am 15 01 2022].
- [24] K. Simonov, „PyYAML,“ 13 10 2021. [Online]. Available: <https://pypi.org/project/PyYAML/>. [Zugriff am 15 01 2022].
- [25] „Python Dictionaries,“ W3Schools, 2022. [Online]. Available: https://www.w3schools.com/python/python_dictionaries.asp. [Zugriff am 15 01 2022].
- [26] „What are people counters and why are they important?,“ Genetec Inc., 2021. [Online]. Available: <https://www.genetec.com/blog/products/what-are-people-counters-and-why-are-they-important>. [Zugriff am 15 01 2022].
- [27] O.-P. Heinisuo, „pypi.org,“ 11 07 2021. [Online]. Available: <https://pypi.org/project/opencv-python/4.5.3.56/>. [Zugriff am 15 01 2022].
- [28] A. Clark, „pypi.org,“ 15 10 2021. [Online]. Available: <https://pypi.org/project/Pillow/8.4.0/>. [Zugriff am 15 01 2022].
- [29] A. Ronacher, „pypi.org,“ 04 10 2021. [Online]. Available: <https://pypi.org/project/Flask/>. [Zugriff am 15 01 2022].
- [30] K. Reitz, „pypi.org,“ 13 07 2021. [Online]. Available: <https://pypi.org/project/requests/2.26.0/>. [Zugriff am 15 01 2022].
- [31] N. Spurrier, T. Kluyver und J. Quast, „pypi.org,“ 21 01 2020. [Online]. Available: <https://pypi.org/project/pexpect/>. [Zugriff am 15 01 2022].
- [32] „Github-espressif,“ espressif, 20 12 2021. [Online]. Available: <https://github.com/espressif/arduino-esp32/tree/master/libraries/BLE/src>. [Zugriff am 17 01 2022].

- [33] „ESP Sleepmodes,“ espressif Systems, 06 01 2022. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/sleep_modes.html. [Zugriff am 17 01 2022].
- [34] „GitHub - Adruit DHT,“ 2021, 25 12 2021. [Online]. Available: <https://github.com/adafruit/DHT-sensor-library>. [Zugriff am 17 01 2022].
- [35] „GitHub - CSS811 Library,“ Adafruit Industries, 02 09 2021. [Online]. Available: https://github.com/adafruit/Adafruit_CCS811. [Zugriff am 17 01 2022].
- [36] „GitHub - MPU6050 Repository,“ Adafruit Industries, 12 01 2022. [Online]. Available: https://github.com/adafruit/Adafruit_MPU6050. [Zugriff am 17 01 2022].
- [37] „GitHub - VL53L0X Repository,“ Adafruit Industries, 21 12 2021. [Online]. Available: https://github.com/adafruit/Adafruit_VL53L0X. [Zugriff am 17 01 2022].
- [38] L. ". Fried, „<https://learn.adafruit.com/>,“ Adafruit, 29 07 2012. [Online]. Available: <https://learn.adafruit.com/dht>. [Zugriff am 19 01 2022].
- [39] „www.sciosense.com,“ Sciosense B.V., [Online]. Available: <https://www.sciosense.com/products/environmental-sensors/ccs811-gas-sensor-solution/>. [Zugriff am 19 01 2022].
- [40] „MPU6050,“ InvenSense, 2022. [Online]. Available: <https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/>. [Zugriff am 17 01 2022].
- [41] „Mouser Electronics - vl53l0x,“ Mouser Electronics, Inc., 04 10 2021. [Online]. Available: <https://www.mouser.de/new/stmicroelectronics/stm-vl53lox-sensor/>. [Zugriff am 17 01 2022].
- [42] M. Akbari, „electropeak.com/,“ 03 2021. [Online]. Available: <https://electropeak.com/learn/esp32-bluetooth-low-energy-ble-on-arduino-ide-tutorial/>. [Zugriff am 08 03 2022].
- [43] „btprodbspecificationrefs.blob.core.windows.net,“ Bluetooth SIG, Inc, 04 03 2022. [Online]. Available: <https://btprodbspecificationrefs.blob.core.windows.net/assigned-values/16-bit%20UUID%20Numbers%20Document.pdf>. [Zugriff am 08 03 2022].
- [44] F. Raschbichler, „<https://www.informatik-aktuell.de>,“ Alkmene Verlag GmbH, 13 06 2017. [Online]. Available: <https://www.informatik-aktuell.de/betrieb/netzwerke/mqtt-leitfaden-zum-protokoll-fuer-das-internet-der-dinge.html>. [Zugriff am 28 02 2022].
- [45] C. S. 02, „<http://docs.oasis-open.org>,“ OASIS, 15 05 2018. [Online]. Available: <http://docs.oasis-open.org/mqtt/mqtt/v5.0/cs02/mqtt-v5.0-cs02.html>. [Zugriff am 28 02 2022].
- [46] „<http://arduino-basics.com>,“ [Online]. Available: <http://arduino-basics.com/arduino-mobil/standalone/bootloader/>. [Zugriff am 24 01 2022].
- [47] R. Want, „os.mbed.com,“ Arm Limited, 11 08 2015. [Online]. Available: https://os.mbed.com/teams/Bluetooth-Low-Energy/code/BLE_EddystoneBeacon_Service/file/dfb7fb5a971b/Eddystone.h. [Zugriff am 31 01 2022].

- [48] M. Ashbridge, N. Khazanie und M. Wandschneider, „Github.com,“ Google, 16 04 2016. [Online]. Available: <https://github.com/google/eddystone/blob/master/protocol-specification.md>. [Zugriff am 31 01 2022].
- [49] J. B. -. D. (2BAdvice), Interviewee, Az.:20220119_Bachelorarbeit_People-Counter/Raumsensorik. [Interview]. 07 02 2022.
- [50] „exp-tech.de,“ EXP GmbH, [Online]. Available: <https://www.exp-tech.de/sensoren/temperatur/4221/dht11-humidity-temperature-sensor-old-version>. [Zugriff am 14 02 2022].
- [51] „www.elektra-wandelt.de,“ [Online]. Available: <https://www.elektra-wandelt.de/ratgeber/Wie-funktionieren-eigentlich-Bewegungsmelder/#:~:text=Passiv%2DInfrarot%2DMelder%20reagieren%20auf,haupts%C3%A4chlich%20auf%20Menschen%20oder%20Tiere..> [Zugriff am 15 02 2022].
- [52] „www.az-delivery.de,“ AZ-Delivery Vertriebs GmbH, 2022. [Online]. Available: <https://www.az-delivery.de/products/bewegungsmelde-modul>. [Zugriff am 15 02 2022].
- [53] „https://www.gesundheit.de,“ FUNKE DIGITAL GmbH, 08 05 2017. [Online]. Available: <https://www.gesundheit.de/familie/freizeit-und-zuhause/gesundes-wohnen-sicheres-zuhause/saubere-luft-gesundes-raumklima>. [Zugriff am 15 02 2022].
- [54] „www.berrybase.de,“ [Online]. Available: <https://www.berrybase.de/Pixelpdfdata/Articlepdf/id/7971/onumber/ADA3566>. [Zugriff am 15 02 2022].
- [55] „www.sedus.com,“ Sedus Stoll AG, 2022. [Online]. Available: <https://www.sedus.com/de/produkte/tische/selab-high-desk>. [Zugriff am 16 02 2022].
- [56] A. A. Jabbaar, „create-arduino.cc,“ 17 09 2019. [Online]. Available: <https://create.arduino.cc/projecthub/abdularbi17/ultrasonic-sensor-hc-sr04-with-arduino-tutorial-327ff6>. [Zugriff am 16 02 2022].
- [57] F. Petit, „https://www.blickfeld.com,“ Blickfeld GmbH, 19 08 2020. [Online]. Available: <https://www.blickfeld.com/de/blog/was-ist-lidar/>. [Zugriff am 25 02 2022].
- [58] „sedus.com,“ Sedus Stoll AG, [Online]. Available: <https://www.sedus.com/de/produkte/workshop-tools-accessoires/selab-boards>. [Zugriff am 20 02 2022].
- [59] „elektronik-kompendium.de,“ [Online]. Available: <https://www.elektronik-kompendium.de/sites/bau/1503041.htm>. [Zugriff am 20 02 2022].
- [60] W. EWALD, „wolles-elektronikkiste.de,“ 27 11 2020. [Online]. Available: <https://wolles-elektronikkiste.de/mpu6050-beschleunigungssensor-und-gyroskop>. [Zugriff am 20 02 2022].
- [61] „www.lernhelper.de,“ Duden Learnattack GmbH, [Online]. Available: <https://www.lernhelper.de/schuelerlexikon/physik-abitur/artikel/piezoelektrischer-effekt#>. [Zugriff am 20 02 2022].

- [62] „components101.com,“ Components101, 24 05 2021. [Online]. Available: <https://components101.com/sensors/801s-vibration-sensor-module-pinout-features-datasheet-working-application-alternative>. [Zugriff am 20 02 2022].
- [63] „sedus.com,“ Sedus Stoll AG, [Online]. Available: <https://www.sedus.com/de/produkte/sitzmoebel/selab-hopper>. [Zugriff am 20 02 2022].
- [64] „wearic.com,“ Wearic, [Online]. Available: <https://www.wearic.com/product/smart-textiles-kit-wit-nano-controller/>. [Zugriff am 20 02 2022].
- [65] „www.wearic.com,“ WEARIC, [Online]. Available: https://www.wearic.com/wp-content/uploads/2018/07/WEARIC_Datasheet_Expansion-Board.pdf. [Zugriff am 20 02 2022].
- [66] D. Geevarghese, „www.cabotsolutions.com,“ Cabot Solutions Inc., 01 02 2018. [Online]. Available: <https://www.cabotsolutions.com/ble-vs-wi-fi-which-is-better-for-iot-product-development>. [Zugriff am 21 02 2022].
- [67] „docs.opencv.org,“ 5 07 2021. [Online]. Available: https://docs.opencv.org/4.5.3/d4/d15/group__videoio__flags__base.html. [Zugriff am 07 03 2022].
- [68] „https://developer.mozilla.org,“ Mozilla, 07 02 2022. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Glossary/Base64>. [Zugriff am 14 03 2022].
- [69] G. Hidalgo, „github.com,“ 23 04 2020. [Online]. Available: https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/v1.7.0/doc/quick_start.md. [Zugriff am 09 03 2022].
- [70] S. Cook, „https://github.com/seancook/,“ 28 01 2022. [Online]. Available: <https://github.com/seancook/docker-openpose-cpu>. [Zugriff am 10 03 2022].
- [71] „www.hivemq.com,“ HiveMQ GmbH, 20 08 2019. [Online]. Available: <https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/>. [Zugriff am 10 03 2022].
- [72] „https://grafana.com,“ [Online]. Available: <https://grafana.com/>. [Zugriff am 14 03 2022].
- [73] B. Dyer, „itsfoss.com,“ 5 06 2021. [Online]. Available: <https://itsfoss.com/linux-daemons/>. [Zugriff am 14 03 2022].
- [74] S. Cope, „steves-internet-guide.com,“ 30 09 2020. [Online]. Available: <http://www.steves-internet-guide.com/mosquitto-tls/>. [Zugriff am 14 03 2022].
- [75] R. Light, „mosquitto.org,“ Eclipse Foundation, Inc., [Online]. Available: <https://mosquitto.org/man/mosquitto-conf-5.html>. [Zugriff am 14 03 2022].
- [76] mstahl, „praxistipps.chip.de,“ BurdaForward GmbH , 30 09 2019. [Online]. Available: https://praxistipps.chip.de/bluetooth-le-low-energy-das-ist-der-unterschied-zu-normalem-bluetooth_114239. [Zugriff am 20 01 2022].

Abbildungsverzeichnis

Abbildung 1- Sedus Möbel der se:lab Modellreihe (Quelle: Sedus Stoll AG, 2020)	3
Abbildung 2 - Bluetooth Low Energy aka Bluetooth Smart (Quelle: wiki.lm-technologies.com)	7
Abbildung 3 - BLE Advertising Data Diagramm (Quelle: electropeak.com)	8
Abbildung 4 - Aufbau GATT (Quelle: electropeak.com)	9
Abbildung 5 - Schaubild MQTT (Quelle: informatik-aktuell.de)	10
Abbildung 6 - MQTT-Topic	10
Abbildung 7 - Datenfluss und Kommunikation	11
Abbildung 8 - Arbeitsweise Sensorik.....	11
Abbildung 9 - Grundaufbau Arduino Sketch	12
Abbildung 10 - Auszug declaration "Chair_Sensor".....	12
Abbildung 11 - Void Loop	13
Abbildung 12 – Definition eines Eddystone BLE Beacon (https://os.mbed.com/ , 2015).....	13
Abbildung 13 - Working Modes im Arduino Sketch	14
Abbildung 14 – Definition von Influencer und Switchmaster in mqtt-to-influx-broker Konfigurationsdatei.....	15
Abbildung 15 - mod_sendData.py	16
Abbildung 16 - Einlesen von Konfigurationsdateien	16
Abbildung 17 - Konfiguration in YAML.....	17
Abbildung 18 - mod_read_config.py	17
Abbildung 19 - Arbeitsweise People-Counter.....	18
Abbildung 20 - Projektordner "PEOPLE_COUNTER_EDGE"	18
Abbildung 21 - OpenPose Working Modes	19
Abbildung 22 - Send Modes.....	19
Abbildung 23 - Datenfluss BLE-MQTT-Gateway.....	20
Abbildung 24- Projektordner "BLE-MQTT-GATEWAY"	20
Abbildung 25 - Komponenten "OpenPose-Server"	21
Abbildung 26 - OpenPose Erkennung.....	21
Abbildung 27 - Projektordner "PEOPLE_COUNTER_SERVER"	22
Abbildung 28- people-counter-server.py - REST-API-EndPoints	22
Abbildung 29 - MQTT-InfluDB-Broker	23
Abbildung 30 - Projektordner "MQTT-INFLUXDD-BROKER"	23
Abbildung 31 - YAML-Konfiguration.....	24
Abbildung 32- Agile Projektfläche mit se:lab (Quelle: sedus.com)	25
Abbildung 33 - ESP32-NodeMCU (Quelle:az-delivery.de)	27
Abbildung 34 - Sensorik für Raumsensor	27
Abbildung 35 - Schaltbild des Raumsensors.....	28
Abbildung 36 - se:lab High Desk (Quelle: sedus.com).....	29
Abbildung 37 - Ultraschall Distanz Sensor (HC-SR04), PIR-Sensor (HC-SR501), LiDAR Distanz Sensor (VL530X)	29
Abbildung 38-Berechnugn der Distanz durch Ultraschall (Quelle: create.arduino.cc).....	30
Abbildung 39 - Schaltbild High-Desk-Sensoren.....	30
Abbildung 40- se:lab Board (Quelle: sedus.com)	31
Abbildung 41 - v.l. Gyrosensor (MPU-6050) und Vibrationssensor (801S)	31
Abbildung 42 - Messprinzip MPU6050 (Quelle: wolles-elektronikkiste.de)	32
Abbildung 43 - Schaltbild se:lab Board Sensor	32
Abbildung 44 - se:lab Hopper (Quelle: sedus.com)	33
Abbildung 45 - Stoffbasierter Drucksensor von Wearic	33
Abbildung 46 - Schaltbild se:lab Hopper Sensor	34
Abbildung 47 - Raspberry Pi 3B+ (Quelle: reichelt.de).....	35

Abbildung 48 - Raspberry Pi 3B+ Spezifikationen (Quelle: raspberrypi.com)	35
Abbildung 49 - Technische Daten ESP32 (Quelle: az-delivery.de).....	36
Abbildung 50 - Implementierung von GAP und GATT in Arduino Sketch	36
Abbildung 51 - Aufbau des Eddystone Beacons in Arduino Sketch.....	37
Abbildung 52 - Main-loop des BLE-to-MQTT Gateways.....	38
Abbildung 53- Daten aus Eddystone Beacon für se:lab Board (Whiteboard).....	38
Abbildung 54 - Gegenüberstellung Beacon Bytes zu Daten aus Abb.51	38
Abbildung 55 - send_mqtt_data	39
Abbildung 56 - übersetzen und senden von Beacon Bytes in MQTT-Topics.....	39
Abbildung 57 - Überschreiben des aktuellen working_mode per BLE	40
Abbildung 58 - implementierung des GATT Services für working_mode überschreibung	40
Abbildung 59 – Skizze People Counter Edge Device.....	41
Abbildung 60 - Ausschnitt aus People Counter Edge Konfiguration.....	41
Abbildung 61 - Aufnahme per Kamera und Konvertierung in Base64.....	42
Abbildung 62 - Übertragung an OpenPose Server.....	43
Abbildung 63- Ergebnisse von OpenPose bei Unterschiedlicher Kamerahöhe (links 1.5m, rechts 2.3m)	43
Abbildung 64 - OpenPose Server Konfiguration	44
Abbildung 65 - OpenPose Ergebnisbild im erstellter Keypoints.json-Datei.....	45
Abbildung 66 - People Counting mit Docker (Methode: count_people_docker)	45
Abbildung 67 - People Counting Docker Befehl	46
Abbildung 68 - OpenPose Keypoints in JSON	46
Abbildung 69- People Counting nativ via Python (Methode: count_people_openpose_native)	47
Abbildung 70 - Kompilieren von OpenPose mit Python Wrapper.....	47
Abbildung 71 - Main Loop von mqtt-to-influx-broker.py (links) mqtt_config Parameter für MQTT-to-Influx-Broker (rechts).....	48
Abbildung 72 - Beispiel Topic Levels (Quelle: www.hivemq.com).....	48
Abbildung 73 – MQTT-Topics (links) Implementierung „on_message“-Callback verkürzt (rechts). Abbildung 74 - Peoplecoutner MQTT-Payload	49
Abbildung 75 - Implementierung sendInfluxData-Methode in mod_SendData.py	50
Abbildung 76 - Schaubild InfluxDB Schema und MQTT-Topic.....	51
Abbildung 77 - InfluxDB-API Token in InfluxDB Web-Frontend.....	51
Abbildung 78 – People Counter Graph im InfluxDB Data Explorer	51
Abbildung 79 - Dashboard in Grafana	52
Abbildung 80 - Installations-Skript und Service-File von people-counter-client	52
Abbildung 81- Verschlüsselte Kommunikation der erarbeiteten Lösung	53
Abbildung 82 - SSL Konfigurationen v.1 Mosquitto und InfluxDB	53
Abbildung 83 - SSL-Config für MQTT und INFLUXDB in Konfigurationsdatei von MQTT-InfluxDB-Broker	54
Abbildung 84 - Mosquitto Benutzerberechtung.....	54
Abbildung 85 - MQTT User Konfiguration für MQTT-Kommunikation.....	54
Abbildung 86 - BLE Kennwort.....	54
Abbildung 87- Grafana-Dashboard Raumnutzung.....	55