Dokumentation zur IT-PROJEKT

**KÜHLRAUM**

**Die Steuerung einer Tür (sowie. Lüfter) eines Kühlraums mit Temperatur- und Feuchtigkeit- Erfassung**

YVAN VALDER SIMO GUENO, 7212803

14. Dezember 2022

Prüfer: Pr. Dr. Kai Luppa

Vorwort

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Quelleverzeichnis

1 Einleitung

Die folgende Projektdokumentation schildert den Ablauf eines IT-Projektes, welches im Rahmen der IT-Projekt 33 ( ITP ) Praktikum durchgeführt wurde. MQTT, Tkinter, Python und Raspberry-Pi (JOY-PI) sind Themen, die selbstverständlich mit diesem IT-Projekt verbunden. Bei JOYPI handelt es sich um einen Raspberry Pi, bei dem alle anschließbaren Komponenten in einer Platine direkt mit dem Raspberry Pi verbunden sind.

* 1. Projektbeschreibung

In unserem ITP, der sich „KÜHLRAUM“ lautet, werden unterschiedliche Sensoren und Aktoren von einer „Internet of Things“ (IoT) Komponente ( hier JOY-PI) definiert.

Als Sensoren werden der Button für den Übergang zur digitalen Steuerung, der Motion-Sensor für das Öffnen der Tür und der DHT11 für die Temperatur- und Feuchtigkeit- Erfassung verwendet.

Als Aktoren haben wir hingegen unseren LED-Matrix für die Darstellungen von Zuständen und für die Anzeige der Temperatur und Feuchtigkeit, mehreren sich nacheinander aufleuchtenden LED zum Zeigen, dass sich die Tür öffnen wird , einen Buzzer , die ein Ton bei jeder Öffnung ergibt, und einen STEP-Motor , der die Tür darstellt.

Die Tür kann nicht nur manuell über den Bewegungssensor sondern auch digital über die grafische Benutzeroberfläche (hier „TKinter“) geöffnet werden, aber der Lüfter kann nur über die Benutzeroberfläche („Graphics User Interface- (GUI) “) bedient werden.

Zur Kommunikation zwischen unserer JOYPI und der GUI wird mit Hilfe eines Brokers und durch einer entsprechenden Software („Kommunikation Protokoll“) darunter „Message Queuing Telemetry Transport“ (MQTT).

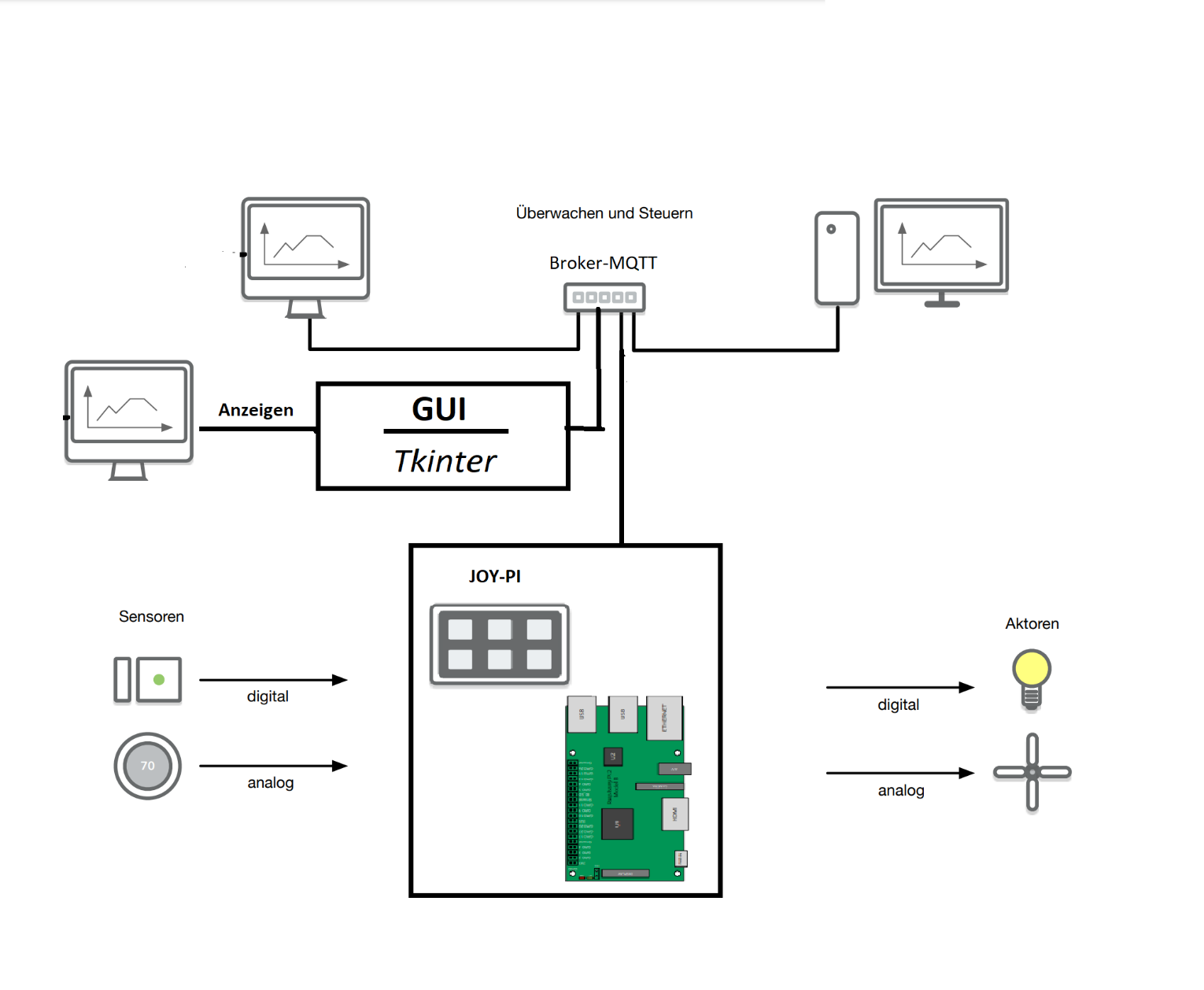


Abbildung 1: Konfiguration des Projekts

* 1. Projektziel

Das Ziel des Projekts ist eine Tür eines Kühlraumes zu steuern (d.h. das Öffnen und Schließen der Tür), wobei bei jedem Öffnen die Temperatur des Kühlraums gemessen wird, und die Steuerung eines Lüfters (d. h. einer Klimaanlage) in Abhängigkeit von der Temperatur des Kühlraums.

Die Verwendungszwecke und die Möglichkeit der Umsetzung sind vielfältig. Es kann für die Kontrolle der Temperatur von Nährstoffen in einer Produktion oder in der Medizin verwendet werden, wenn man z. B. die Temperatur von Leichen und bestimmten Medikamenten wie Insulin kontrollieren möchte. Es kann auch eingesetzt werden, um die Ein- und Ausgänge von Mitarbeitern zu kontrollieren, Arbeitszeiten zu erfassen (bei Einsatz der Chip-Komponente)

* 1. Projektabgrenzung

Das Projekt kann nicht direkt manuell und über die Schnittstelle gesteuert werden, d. h. die Tür kann nur dann über die Schnittstelle gesteuert werden, wenn die digitale Steuerung mit der Schaltfläche aktiviert wurde.

Es werden zwei Codes benötigt, einer für die grafische Benutzeroberfläche und einer für die Komponenten, da einige Python-Bibliotheken nur auf dem Raspberry Pi und nicht auf dem PC funktionieren und umgekehrt (z. B. ist Tkinter nur auf dem PC nutzbar, während DHT11(Temperatur und Luftfeuchtigkeit) nur auf dem RPI verwendet wird).

Außerdem schaltet sich der Lüfter nicht direkt bei höchster oder tiefster Temperatur ein und hat keine definierte Zykluszeitraum. Im Fall der Benutzeroberfläche wartet ein Button, der gedrückt wird (z. B. open), nicht darauf, dass die Funktion des zuletzt gedrückten Buttons vollständig ausgeführt wird, bevor er seinen Status sendet und darstellt.

1. Projektplanung

2.1 Projektstart:

Für die Projektplanung und Umsetzung standen dem Autor 3 Monate lang zur Verfügung. Diese wurden vor Projektbeginn auf verschiedene Phasen verteilt, die während der Softwareentwicklung durchlaufen werden. Die Projektphase umfasst eine Erstellung von Anforderungsanalyse, die Architektur- und Design- Phase ,die Implementierung der Hardware-Komponenten, die Implementierung der GUI , die Integration von GUI und Hardware-komponenten , eine Testphase und eine Dokumentation. Eine Gannt

Diagramm sowie die Hauptphasen lassen sich der Tabelle 1: Gannt Diagramm entnehmen.

Für dieses Projekt sind alle Ressourcen aufgelistet, die für das Projekt eingesetzt wurden. Damit sind Hard und Software Ressourcen gemeint. Bei der Auswahl der verwendeten Software wurde darauf geachtet, dass diese kostenfrei zur Verfügung stehen. bei unserem Fall ist alles schon im unserem JoyPI zur Verfügung.

Anforderungsanalyse:

Die Anforderungen an das Projekt waren die Verbindung einer JOY-PI als IoT-Komponente, die Verwendung von MQTT als Kommunikationsprotokoll Komma die Verwendung von Python und Tkinter für die Entwicklung der Anwendung, die Verwendung von Sensoren wie Button, Motion-Sensor und DHT11 Komma die Verwendung von Aktoren wie LED-Matrix, Leds, Buzzer und STEP-Motor Und die Erstellung einer grafischen Benutzeroberfläche (GUI) zur Steuerung der Komponenten.

Es handelte sich um eine Anwendung zur Steuerung und Überwachung von verschiedenen Komponenten in einen Kühlraum, die mit Hilfe einer Raspberry Pi (JOY-Pi) und verschiedener Sensoren und Aktoren umgesetzt wird. Die Anwendung kommuniziert mit einer grafischen Benutzeroberfläche (GUI) über das MQTT-Protokoll und wurde in Python und Tkinter entwickelt.

Als Risiken waren unsere fehlende Erfahrung mit der Verwendung von Mqtt und Tkinter , die Schwierigkeiten bei der Integration von Hardware-Komponenten und die Kompatibilitätsprobleme mit der JOY-PI oder den verwendeten Sensoren und Aktoren

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **KÜHLRAUM** | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  |
| *Wählen Sie rechts einen Zeitraum zum Hervorheben aus. Es folgt eine Legende, die das Diagramm beschreibt.* | | | | | **Hervorgehobener Zeitraum:** | 15 |  |  | Dauer des Plans | | | | |  | Tatsächlicher Start | | | |  | % abgeschlossen | | | | |  | | Tatsächlich (hinter dem Plan) | | | | | | | |  | | % abgeschlossen (hinter dem Plan) | | | | | | | |
| **AKTIVITÄT** | **START DES PLANS** | **DAUER DES PLANS** | **TATSÄCHLICHER START** | **TATSÄCHLICHE DAUER** | **PROZENT ABGESCHLOSSEN** | **ZEITRÄUME** | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | | **20** | | **21** | **22** | **23** | **24** | **25** | **26** | **27** | | **28** | | **29** | **30** | **31** | **32** | **33** | **34** | **35** |
| **Erstellung von Anforderungsanalyse** | 1 | 2 | 1 | 2 | **100%** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  |
| **Architektur- und Design-Phase** | 2 | 2 | 2 | 3 | **100%** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  |
| **Erstellung von Projektplan** | 2 | 2 | 2 | 2 | **100%** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  |
| **Implementierung der Hardware-Komponenten** | 3 | 5 | 3 | 4 | **100%** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  |
| **Implementierung der GUI** | 6 | 4 | 6 | 3 | **100%** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  |
| **Integration von GUI und Hardware-Komponenten** | 7 | 3 | 7 | 2 | **100%** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  |
| **Testphase** | 8 | 4 | 8 | 3 | **90%** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  |
| **Dokumentation** | 7 | 6 | 8 | 5 | **20%** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  | |  | |  |  |  |  |  |  |  |

Abbildung 2: Gannt-Diagram für die Zeitliche verlauf des Projekt

Implementierung der Hardware-komponenten

Diese beinhaltet das physische Zusammenbauen und Verbinden der Geräte gemäß den spezifischen Anforderungen des Projekts.

Für die Implantierung der Hardware-Komponenten in dem JOYPI, gab es 3 Möglichkeiten : Ersten der JOYPI mit VNC Viewer zu verbinden, zweiten direkt auf dem JOYPI mithilfe einer Tastatur oder dritten den Thonny (eine [integrierte Entwicklungsumgebung](https://en.wikipedia.org/wiki/Integrated_development_environment)- IDE für [Python](https://en.wikipedia.org/wiki/Python_(programming_language))) direkt mit JOYPI zu verbinden .

Für diesen Projekt haben wir mit VNC gearbeitet (Sieh Abbildung 3). Der VNC (Virtual Network Computing) ist eine [Software](https://de.wikipedia.org/wiki/Software), die den [Bildschirminhalt](https://de.wikipedia.org/wiki/Bildschirm) eines entfernten [Rechners](https://de.wikipedia.org/wiki/Computer) ([Server](https://de.wikipedia.org/wiki/Server)) auf einem lokalen Rechner ([Client](https://de.wikipedia.org/wiki/Client)) anzeigt und im Gegenzug [Tastatur](https://de.wikipedia.org/wiki/Tastatur)- und [Mausbewegungen](https://de.wikipedia.org/wiki/Maus_(Computer)) des lokalen Rechners an den entfernten Rechner sendet.. Die folgende Links stell seine Konfiguration für den JOYPI dar. ([VNC-Viewer herunterladen](https://www.realvnc.com/en/connect/download/viewer/raspberrypi/) und [VNC Connect](https://www.youtube.com/watch?v=21b1GMEnkuM))

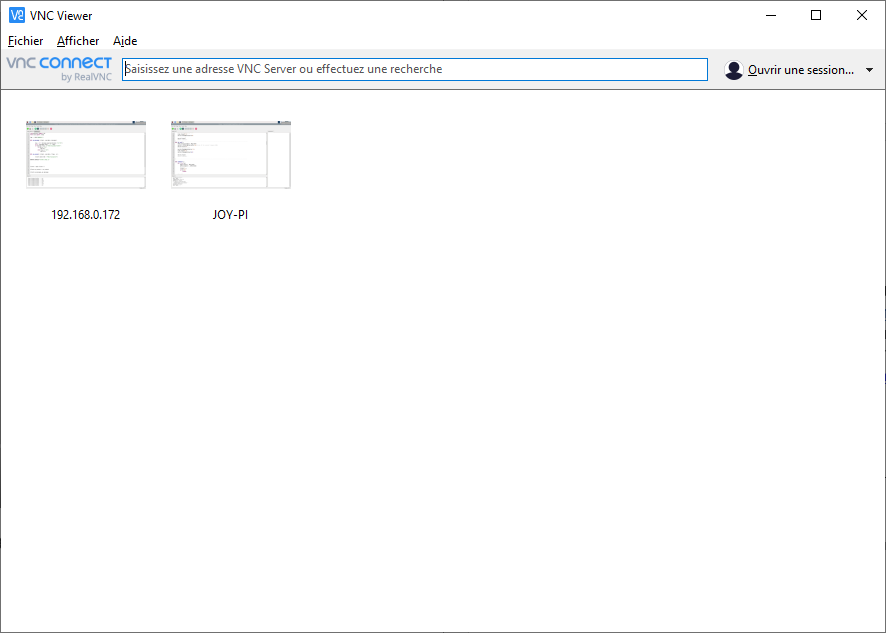


Abbildung 3: VNC-Viewer , der unserer JOYPI und ein anderes Gerät darstellt.

Nach der Verbindung sollten wir im Hauptmain der Raspberry PI landen.

Une image contenant texte, crépuscule, extérieur, établissement

Description générée automatiquement

Abbildung 4: Screenshot der VNC nach der Verbindung mit dem JOYPI

In der Raspberry wird einen Hauptprogramm geschrieben (siehe Anhang ‚final.py‘), um die Geräte zu konfigurieren und zu nutzen.

In diesem Programm muss ich erstmal die Bibliotheken zum Verbinden und steuern von Sensoren und Aktoren importiert, nachdem die Bibliotheken zur Kommunikation mit dem Broker importiert werden.

#Importierung der Bibliothek paho.mqtt.client (mqtt), damit MQTT-clients #miteinander kommuniziert

import paho.mqtt.client as mqtt

#Importierung der Bibliothek zum verbinden und steuern von Sensoren und Aktoren

import RPi.GPIO as GPIO

Außerdem muss ich noch Bibliotheken zur Steuerung des Feuchtigkeit und Temperatur („ import dht11“) , und zur Steuerung des Anzeigens des LEDs-Matrix ( „import luma“). Und noch dazu, müssen jede Elemente durch seines Pins definiert werden.

Die GPIO „Boardkonfiguration“ wird als Nutzung Mode unserer Pins benutzt („ GPIO.setmode(GPIO.BOARD) “) und diese Pins müssen noch mal konfiguriert werden ( GPIO.setup(`Pin´, ´GPIO.OUT/GPIO.IN´) .Die Pins für Sensoren werden in „GPIO.IN“ konfiguriert und die Pins für Aktoren in „GPIO.OUT“ konfiguriert.

Auf der Joy-Pi-Platine befinden sich zwei Schalteinheiten mit jeweils 8 Schaltern, die es ermöglichen, zwischen verschiedenen Sensoren und Modulen zu wechseln. Da der Raspberry Pi nur über eine begrenzte Anzahl von GPIO-Pins verfügt, sind diese Schalter notwendig, um mehr Sensoren und Module als die Anzahl der verfügbaren GPIO-Pins zu verwenden.

In unserem Projekt wurde den PinNo37 link im 7. Schalter für unseren Button zum Übergang zur digitalen Steuerung aktiviert (dieser Pin ist auch mit dem LED 37 angeschlossen ) und den PinNo22 recht im 8. Schalter für den Servo Motor.(Der Servo Motor kann entweder PinNo22 für servo2 oder PinNo37 für Servo1 sein und die Beiden haben unterschiedliche Anschlüsse)

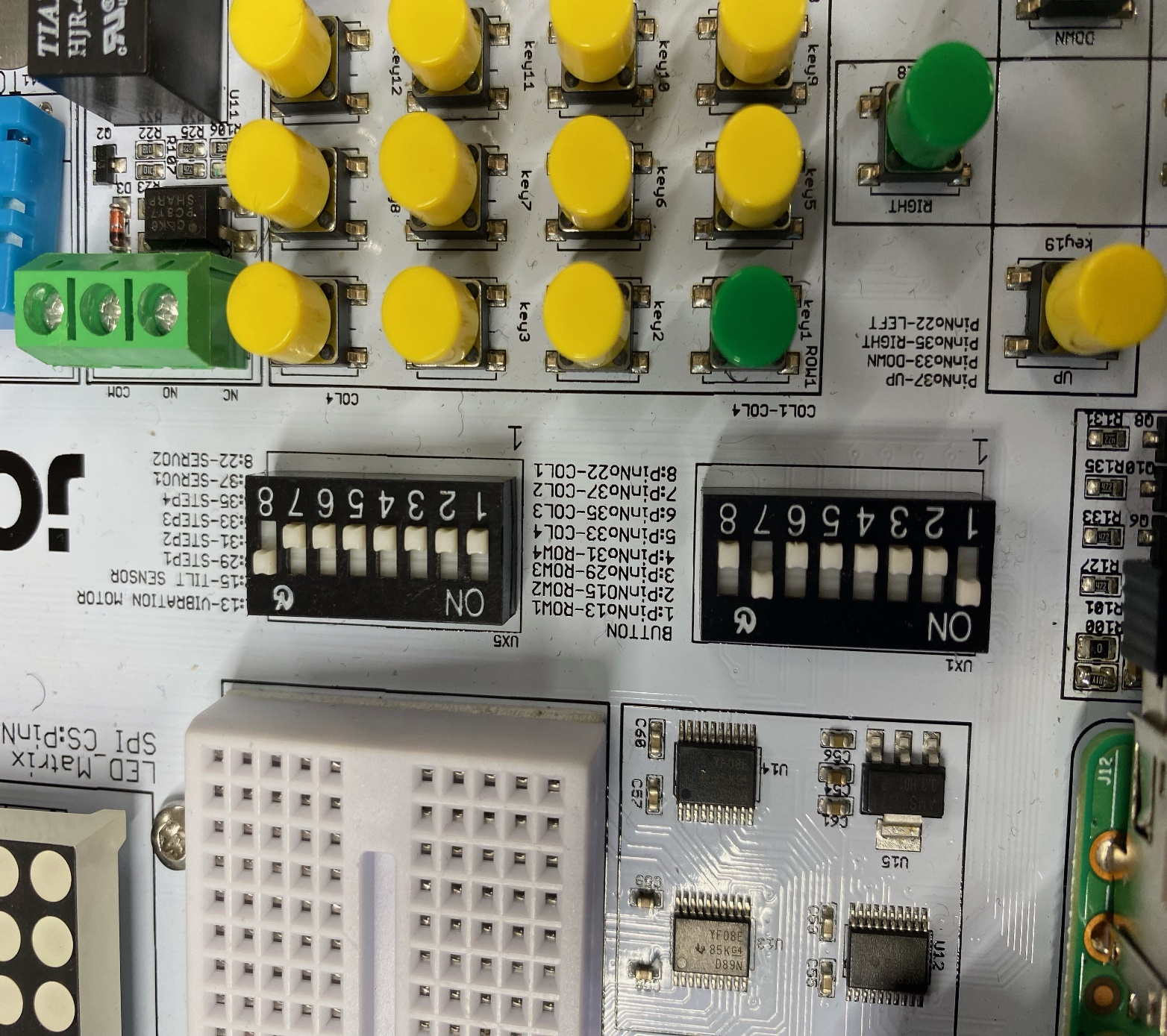


Abbildung 5:Schalter-Zum umschalten zwischen Sensoren und Modulen

Probleme und Lösung bei dem Hauptprogramm

In der Tat kann die Bedienfunktion der Motion Sensor nicht mit einer anderen Funktion (*hier, der on\_message () für den Broker*) unterbrochen werden, weil der Bewegungssensor als Bedingung seiner While-schleife den Stand eines Buttons hat, damit es unendlich läuft, wohingegen die Schleife *Client. loop\_forever* () der Funktion *on\_message()* von der Nachricht des Brokers(Topics) abhängt.

z.B: def motion():

while GPIO.input(interrupt)==GPIO.HIGH or msg!=‘Digital‘:

#msg ist ein lokale Variable der Funktion *on\_message*(), und kann nur #in der Funktion motion() mithilfe eines Pointers benutzt werden.

**Der Hauptprogramm**

#Alle definierten Bibliotheken müssen installiert werden

#---------------------------------------------------------------------------------

----------------------------------------------------------------------------------

#Importierung der Bibliothek paho.mqtt.client (mqtt), damit MQTT-clients #miteinander kommuniziert

import paho.mqtt.client as mqtt

#Importierung der Bibliothek zum verbinden und steuern von Sensoren und Aktoren

import RPi.GPIO as GPIO

#Importierung der Funktion , damit die While-Schleife gleichzeitig mit dem #loop\_forever funktioniert

import threading

#Importierung der Bibliothek für die Zeit und für den Method sleep()

import time

from time import sleep

#Importierung der Biliothek zur Steuerung des Feutchtigkeitssensors sowie der #Temperatur

import dht11

#bibliothek luma und luma.led\_matrix , um einen Text auf einem LED-Matrix-Display #anzuzeigen

import luma

from luma.led\_matrix.device import max7219

from luma.core.interface.serial import spi, noop

from luma.core.render import canvas

from luma.core.virtual import viewport

from luma.core.legacy import text, show\_message

from luma.core.legacy.font import proportional, CP437\_FONT, TINY\_FONT, SINCLAIR\_FONT, LCD\_FONT

#---------------------------------------------------------------------------------

led =[29,31,33,35,38] #Pins led

interrupt=22 #Button\_pin für Tkinter

b= 12 #Buzzer Pin

relay\_pin = 40 #relay Pin

motion\_pin = 16 #Pin des Bewegungssensors

servoPin = 37 #Servo Pin

DHT11\_pin= 7 #DHT11 Pin

#---------------------------------------------------------------------------------

GPIO.setmode(GPIO.BOARD) #Die GPIO Boardkonfiguration benutzen.

GPIO.setwarnings(False) #Deaktivieren der Anzeige von GPIO-Warnungen

GPIO.setup(relay\_pin, GPIO.OUT) #konfiguration des Relay\_pins als Ausgang

GPIO.setup(led, GPIO.OUT) #konfiguration der LEDs als Ausgang

GPIO.setup(b, GPIO.OUT) #Konfiguration des Buzzers als Ausgang

GPIO.setup(servoPin, GPIO.OUT) # Set servoPin to OUTPUT mode

GPIO.setup(motion\_pin, GPIO.IN) #Der Pin der Deklarierten Variable wird als #Input(Eingang) gesetzt.

GPIO.setup(interrupt, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP) #Hier wird den Pin als

#Eingang gesetzt.

#---------------------------------------------------------------------------------

#--------------Funktion zum automatischen Öffnen und Schließen der Tür -----------

def tür():

GPIO.setup(servoPin, GPIO.OUT)

# pin 37 for servo1 frequenz 50Hz

servo1 = GPIO.PWM(servoPin,50)

# 0° wird für den Servo der Referenz Position sein

servo1.start(0)

# Turn servo1 to 90°

servo1.ChangeDutyCycle(7.5)

time.sleep(0.5)

servo1.ChangeDutyCycle(0)

# Wait for 2 seconds

time.sleep(3)

# servo1 back to 0°

servo1.ChangeDutyCycle(2)

time.sleep(0.5)

servo1.ChangeDutyCycle(0)

servo1.stop()

#GPIO.cleanup()

#------------------Funktion nur zum Schließen der Tür-----------------------------

def tür\_zu():

GPIO.setup(servoPin, GPIO.OUT)

# pin 37 for servo1 frequenz 50Hz

servo1 = GPIO.PWM(servoPin,50)

# 0° wird für den servo der Referenz position sein

servo1.start(0)

# servo1 back to 0°

servo1.ChangeDutyCycle(2)

time.sleep(0.5)

servo1.ChangeDutyCycle(0)

servo1.stop()

#GPIO.cleanup()

#------------------Funktion nur zum Öffnen der Tür----------------------------

def tür\_auf():

GPIO.setup(servoPin, GPIO.OUT)

# pin 37 for servo1 frequenz 50Hz

servo1 = GPIO.PWM(servoPin,50)

# 0° wird für den servo der Referenz position sein

servo1.start(0)

# Turn servo1 to 90

servo1.ChangeDutyCycle(7.5)

time.sleep(0.5)

servo1.ChangeDutyCycle(0)

servo1.stop()

#GPIO.cleanup()

#-----------------------Ankündigung oder Laden der Türöffnung---------------------

def ledwater():

for i in led:

GPIO.setup(i, GPIO.OUT)

GPIO.output(i , GPIO.LOW)

#led on

sleep(0.5)

if i==38:

break;

#led off

GPIO.output(led , GPIO.HIGH)

#---------------Funktion zum Abrufen der temperature und Feuchtigkeit-------------

def temp():

# Temperatur und Feuchtigkeit mit DHT11 abrufen

instance = dht11.DHT11(DHT11\_pin)

result = instance.read()

#Ruf Datein bis gültige Werte

while not result.is\_valid():

result = instance.read()

#Anzeigen der Result in der Konsole

print("Temperature: %-3.1f C" % result.temperature)

print("Humidity: %-3.1f %%" % result.humidity)

return result

#-----Funktion zum anzeigen der Temperatur und Feuchtigkeit auf dem Matrix -------

def matrix(result):

# Matrix Gerät festlegen und erstellen.

serial = spi(port=0, device=1, gpio=noop())

device = max7219(serial, cascaded= 1, block\_orientation=90,rotate= 0)

# Matrix Initialisierung in der Konsole anzeigen

print("[-] Matrix initialized")

#conversion der Abgerufenen Werten im String, damit diese darstellbar seien

matrix0=str(result.temperature)

matrix2=str(result.humidity)

#Die Temperatur und feuchtigkeit wird hier zu der GUI geschickt

client.publish("fhdo/itp/gp1/11",matrix0)

client.publish("fhdo/itp/gp1/13",matrix2)

global matrix1

matrix1= "Temperature "+matrix0 +"C"

# Ausgegebenen Text in der Konsole Anzeigen

print("--Temperature: %s Grad --" % matrix0)

#Anzeigen der Temperatur

show\_message(device, matrix1 , fill="white", font=proportional(CP437\_FONT), scroll\_delay=0.1)

#----------------------Funktion bei der Öffnung der Tür---------------------------

def on\_open():

global open\_1

open\_1='Door Open.....'

#veröffebtlichung der Stand der Tür auf der grafischen Benutzeroberfläche

client.publish("fhdo/itp/gp1/12",'op')

print(open\_1)

#Buzzer Ton

#Gebe Geraeusch aus

GPIO.output(b, GPIO.HIGH)

#warte eine halbe Sekunde

time.sleep(0.5)

#Stoppe Geraeuschausgabe

GPIO.output(b, GPIO.LOW)

#Led anzeigen

ledwater()

# Oeffne Relais

GPIO.output(relay\_pin, GPIO.LOW)

# warte eine halbe Sekunde

time.sleep(0.5)

# schliesse Relais

GPIO.output(relay\_pin, GPIO.HIGH)

# Wird der print Befehl ausgeführt

time.sleep(0.1)

# 0,1 Sekunde Warten, dann Öffne der Tür

tür()

#---------------------------------------------------------------------------------

#--------------------------Ton bei dem Schließen der Tür mit der GUI-------------

def buzzer\_0():

#Buzzer Ton

GPIO.output(b, GPIO.HIGH)

#Gebe Geraeusch aus

time.sleep(0.25)

#warte eine halbe Sekunde

GPIO.output(b, GPIO.LOW)

#Stoppe Geraeuschausgabe

time.sleep(0.25)

#Buzzer Ton

GPIO.output(b, GPIO.HIGH)

#Gebe Geraeusch aus

time.sleep(0.5)

#warte eine halbe Sekunde

GPIO.output(b, GPIO.LOW)

#Stoppe Geraeuschausgabe

#-------------------------Ton beim Einschalten des Lüfters------------------------

def buzzer\_1():

#Buzzer Ton

GPIO.output(b, GPIO.HIGH)

#Gebe Geraeusch aus

time.sleep(0.25)

#warte eine halbe Sekunde

GPIO.output(b, GPIO.LOW)

#Stoppe Geraeuschausgabe

time.sleep(0.25)

#Buzzer Ton

GPIO.output(b, GPIO.HIGH)

#Gebe Geraeusch aus

time.sleep(0.25)

#warte eine halbe Sekunde

GPIO.output(b, GPIO.LOW)

#Stoppe Geraeuschausgabe

#----------------------------------Ton beim Ausschalten des Lüfters---------------

def buzzer\_2():

#Buzzer Ton

GPIO.output(b, GPIO.HIGH)

#Gebe Geraeusch aus

time.sleep(0.25)

#warte eine halbe Sekunde

GPIO.output(b, GPIO.LOW)

#Stoppe Geraeuschausgabe

time.sleep(0.25)

#Buzzer Ton

GPIO.output(b, GPIO.HIGH)

#Gebe Geraeusch aus

time.sleep(0.25)

#warte eine halbe Sekunde

GPIO.output(b, GPIO.LOW)

#Stoppe Geraeuschausgabe

time.sleep(0.25)

#Buzzer Ton

GPIO.output(b, GPIO.HIGH)

#Gebe Geraeusch aus

time.sleep(0.5)

#warte eine halbe Sekunde

GPIO.output(b, GPIO.LOW)

#Stoppe Geraeuschausgabe

#----------------------------------Funktion zur Steurerung des Lüfters------------

def Turn\_on():

buzzer\_1()

serial = spi(port=0, device=1, gpio=noop())

device = max7219(serial, cascaded= 1, block\_orientation=90, rotate= 0)

show\_message(device, 'On' , fill="white", font=proportional(CP437\_FONT), scroll\_delay=0.7)

def Turn\_off():

buzzer\_2()

serial = spi(port=0, device=1, gpio=noop())

device = max7219(serial, cascaded= 1, block\_orientation=90,

rotate= 0)

show\_message( device, 'Off' , fill="white", font=proportional(CP437\_FONT), scroll\_delay=0.8)

#---------------------Funktion beim Öffnen der Tür mit dem GUI -------------------

def on\_open\_msg():

global open\_1

open\_1='Door Open.....'

print(open\_1)

#Buzzer Ton

GPIO.output(b, GPIO.HIGH)

#Gebe Geraeusch aus

time.sleep(0.5)

#warte eine halbe Sekunde

GPIO.output(b, GPIO.LOW)

#Stoppe Geraeuschausgabe

ledwater()

# Oeffne Relais

GPIO.output(relay\_pin, GPIO.LOW)

# warte eine halbe Sekunde

time.sleep(0.5)

# schliesse Relais

GPIO.output(relay\_pin, GPIO.HIGH)

# Wird der print Befehl ausgeführt

time.sleep(0.1)

# 0,1 Sekunde Warten

# Beginn einer Schleife

tür\_auf()

#--------------------------------Funktion Zur Nutzung des Bewegungsensors---------

#Die Empfindlichkeit des Bewegungssensors kann mit einem Potentiometer 3a #im #Abbildung ++ eingestellt werden.

def motion():

#GPIO.setup(interrupt, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

client.publish("fhdo/itp/gp1/12",’wait’)

while GPIO.input(interrupt)==GPIO.HIGH :

if(GPIO.input(motion\_pin) == 1): # Wenn der Sensor Input = 1 ist

print ("Bewegung Erkannt!") # Wird der print Befehl ausgeführt

on\_open() #Automasches Öffnen und Schließen der Tür

result = temp()

matrix(result) #Darstellung der Temperatur und Feuchtigkeit

time.sleep(0.15) # 0.15 Sekunde Warten

client.publish("fhdo/itp/gp1/12",’wait’)

#if(GPIO.input(motion\_pin) == 0): # Wenn der Sensor Input = 0 ist

#print ("Keine Bewegung ...") # Wird der print Befehl ausgeführt

global a

a="Die manuelle Steuerung ist aus!"

print(a)

#Die Steuerung wird hier nur durch dem GUI gemacht

client.publish("fhdo/itp/gp1/12","ANALOG")

GPIO.cleanup(servoPin)

# der servopin muss nach jeder Drehung gelöscht werden ,damit er immer die #gleiche Drehung macht.

#-----------------------------------------------------------------------------

#--Funktion , wenn der Broker mit der vorgegebenen Topic eine Nachricht bekommt---

def on\_message (client, userdata, message):

msg = str (message.payload.decode ("utf-8"))

print(message.topic,'-->' ,msg)

if message.topic=="fhdo/itp/gp1/12":

if msg=="open":

on\_open\_msg()

result=temp()

matrix(result)

elif msg=='closed':

buzzer\_0()

tür\_zu()

elif msg=='turn':

Turn\_on()

elif msg=='off':

Turn\_off()

elif msg=="press":

print('Pressed...')

thread = threading.Thread(target=motion)

thread.start()

#---------------------Funktion beim Anschließen oder Abonnement der Client ------

def on\_connect (client, userdata, flags, rc):

client.subscribe ('fhdo/itp/gp1/#')

#----------------------------Anwendung der MQTT- Protokoll--------------------------------------------------------------------------------

BROKER\_ADDRESS="broker.emqx.io"

#Objekt der Classe Client wird erstellt

client = mqtt.Client ()

#subskription der Client

client.on\_connect = on\_connect

client.on\_message =on\_message

#Anschließen mit einem Broker

client. connect (BROKER\_ADDRESS)

print ("Connected to MQTT Broker:"+" broker.emqx.io")

#Aufruf der Funktion Motion zum automatischen Öffnen und Schließen der Tür

motion()

client. loop\_forever()

#End

Ohne Thread wird die Funktion "motion()" nur Daten veröffentlichen, nachdem ihre while-Schleife beendet wurde, weil beim Veröffentlichen von "press" die Schleife "client.Loop()" von der while-Schleife der Funktion "motion()" angehalten wird, was das Veröffentlichen der Daten verhindert.

Dafür muss man die threading-Bibliothek Importieren

#Importierung der Funktion , damit die While-Schleife gleichzeitig mit dem #loop\_forever funktioniert

import threading

def on\_message (client, userdata, message):

msg = str (message.payload.decode ("utf-8"))

print(message.topic,'-->' ,msg)

if message.topic=="fhdo/itp/gp1/12":

if msg=="open":

on\_open\_msg()

result=temp()

matrix(result)

elif msg=='closed':

buzzer\_0()

tür\_zu()

elif msg=='turn':

Turn\_on()

elif msg=='off':

Turn\_off()

elif msg=="press":

print('Pressed...')

thread = threading.Thread(target=motion)

thread.start()

Thread

Ein Thread oder Ausführungsstrang ist eine Ausführungseinheit in einem Computerprogramm. Das heißt, ein Thread ist ein Teil eines Programms, der unabhängig und gleichzeitig mit anderen Teilen des Programms ausgeführt werden kann.

Threads sind nützlich für Programme, die mehrere Aufgaben gleichzeitig ausführen müssen, da sie es einem Programm ermöglichen, mehrere Teile gleichzeitig auszuführen. Dies kann die Leistung eines Programms verbessern, da die Systemressourcen effizienter genutzt werden und mehrere Aufgaben parallel ausgeführt werden können.

In Python kann man Threads mithilfe des Moduls threading erstellen. Man kann einen Thread erstellen, indem man eine Klasse definiert, die von threading.Thread erbt, und die Methode "run()" überlädt, die den Code enthält, der im Thread ausgeführt werden soll. Sobald die Klasse definiert ist, kann man einen Thread erstellen, indem man ein Objekt dieser Klasse erzeugt und seine Methode "start()" aufruft.

Liste der Komponenten

1. Servo Motor
2. DHT11
3. Motion Sensor

3a. Potentiometer zur Steuerung der Sensitivität des Motion Sensors

1. LED-Matrix
2. LEDs
3. Button für die digitale Steuerung (d.h. zu dem Abbruch des Motion Sensors)
4. Buzzer
5. Relay

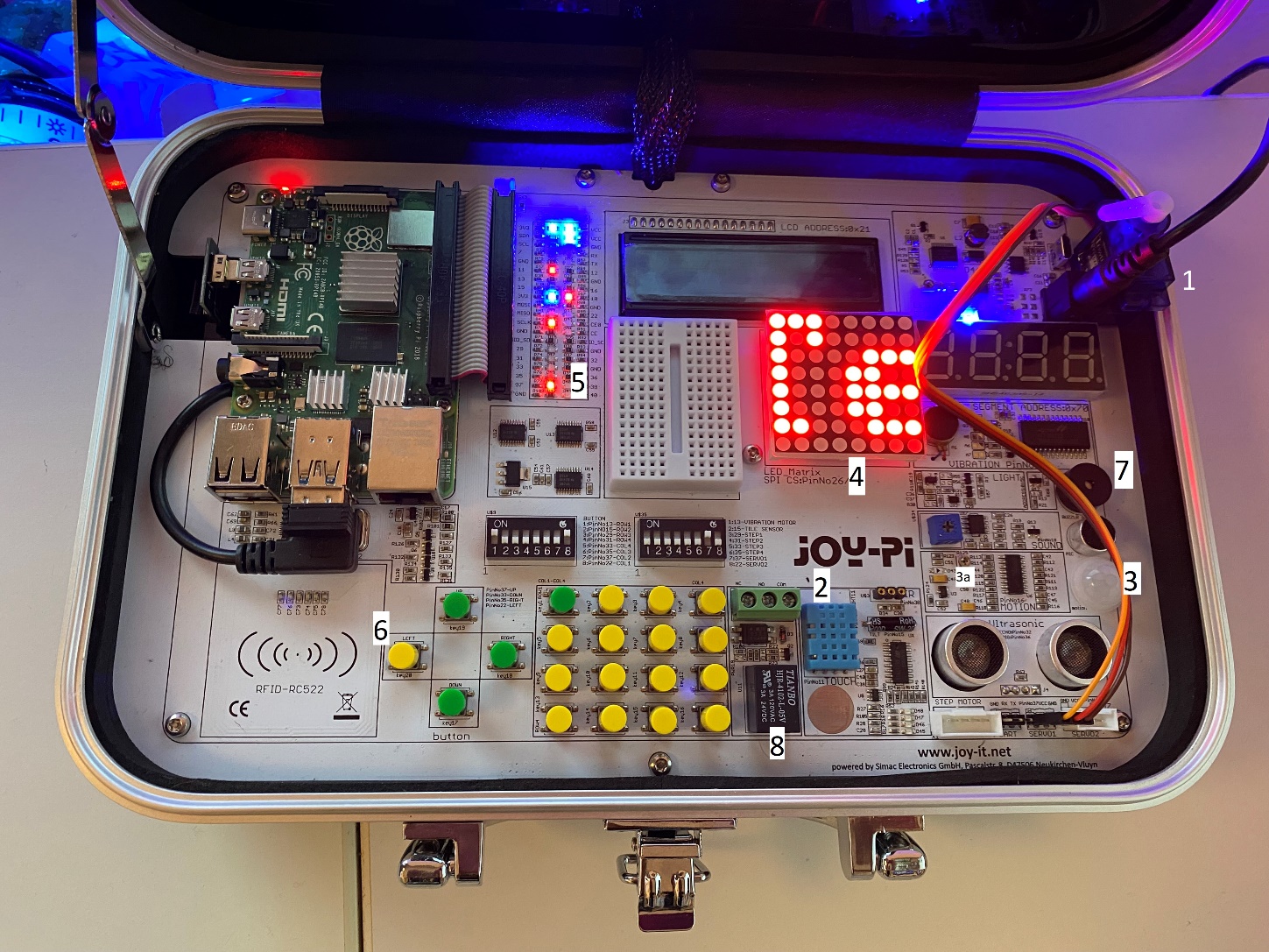


Abbildung 6: JOY-PI und Komponenten des Projekts

Kommunikation zwischen der GUI und der IoT-Komponente: MQTT-Broker

In diesem Projekt werden nur eine Kommunikationsprotokolle verwendet, um die GUI mit der IoT-Komponente zu verbinden. Nämlich das MQTT-Protokoll, das über einen MQTT-Broker verfügbar ist. Der MQTT-Broker dient als Switching Hub und ermöglicht es dem Client (GUI), sich mit der IoT-Komponente zu verbinden und Nachrichten auszutauschen. Das MQTT-Protokoll wird normalerweise mithilfe spezieller Bibliotheken und APIs implementiert, die in Ihren Python-Code integriert sind.

#Importierung der Bibliothek paho.mqtt.client (mqtt), damit MQTT-clients #miteinander kommuniziert

import paho.mqtt.client as mqtt

#---------------------Funktion beim Anschließen oder Abonnement der Client ------

def on\_connect (client, userdata, flags, rc):

client.subscribe ('fhdo/itp/gp1/#')

#----------------------------Anwendung der MQTT- Protokoll--------------------------------------------------------------------------------

BROKER\_ADDRESS="broker.emqx.io"

#Objekt der Classe Client wird erstellt

client = mqtt.Client ()

#subskription der Client

client.on\_connect = on\_connect

client.on\_message =on\_message

#Anschließen mit einem Broker

client. connect (BROKER\_ADDRESS)

print ("Connected to MQTT Broker:"+" broker.emqx.io")

#Aufruf der Funktion Motion zum automatischen Öffnen und Schließen der Tür

motion()

client. loop\_forever()

#End

Um unsere MQTT-Kommunikation zu verfolgen und zu testen, wurde MQTT-Explorer verwendet. Dies ist ein Werkzeug, der uns ermöglicht, MQTT-Nachrichten sofort zu senden und zu empfangen, ohne dass spezielle Client-Software oder Code geschrieben werden muss. MQTT Explorer bietet auch die Möglichkeit, MQTT-Nachrichten zu veröffentlichen und zu abonnieren(subskribieren), um die Funktionalität des MQTT-Systems zu überwachen und zu verstehen. Die Benutzeroberfläche von MQTT Explorer ist intuitiv und einfach zu bedienen, was dieses Programm zu einer guten Wahl für Entwickler macht, die sich mit MQTT-Kommunikation befassen.

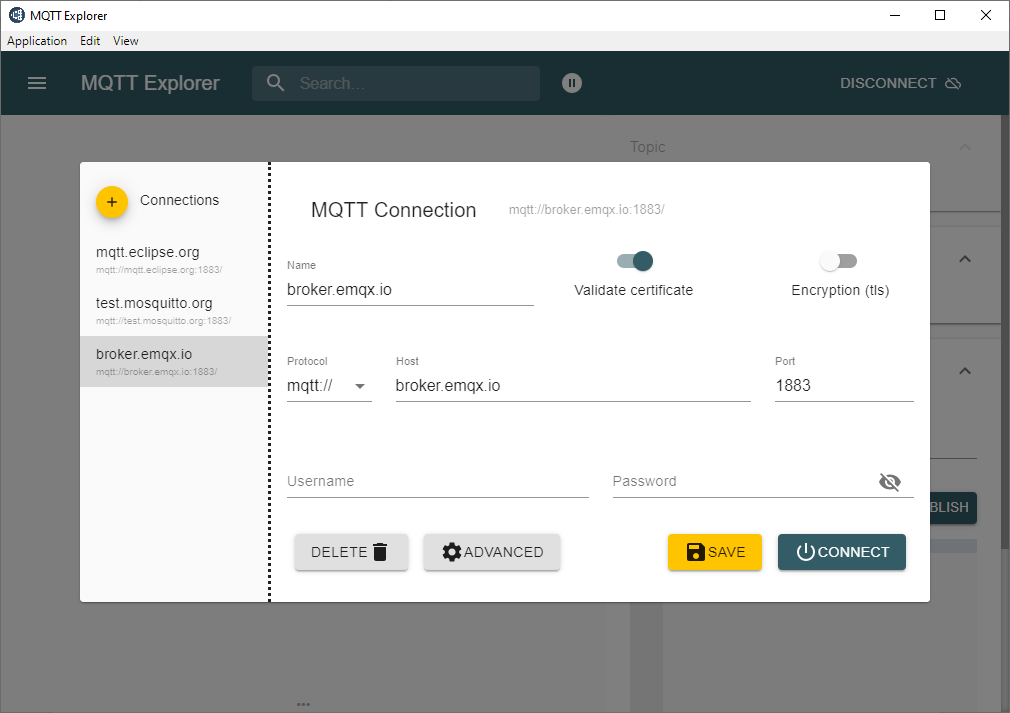


Abbildung 7: Windows-App zur Verfolgung der MQTT-Kommunikation

In Advance-Einstellung muss erstmal in der Subskription in der folgenden Topic „fhdo/itp/gp1/#“ eingegeben werden.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

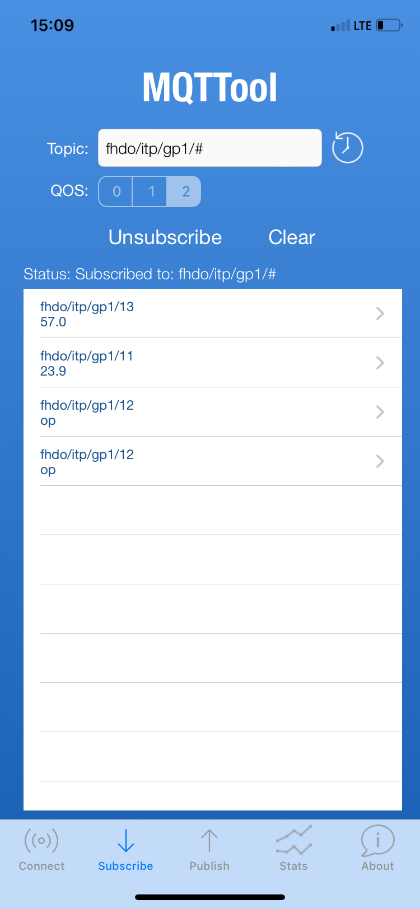
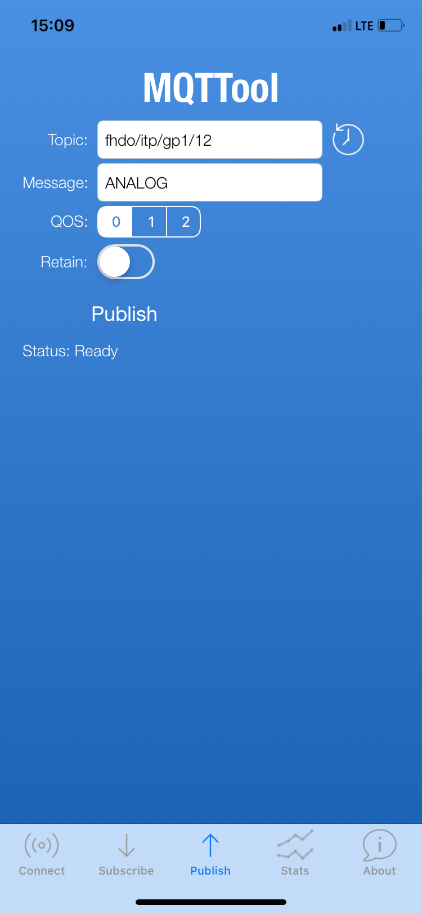
Abbildung 8: Windows-App zur Verfolgung der MQTT-Kommunikation

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 9: Kommandozeile zur Darstellung der laufenden geschickten Nachrichten

[MQTTool](https://apps.apple.com/us/app/mqttool/id1085976398) (Apple Store)

  Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 9: Handys-App zur Verfolgung der MQTT-Kommunikation

Implementierung und Integration der GUI

Die Implementierung der GUI in das Projekt könnte in Python mit verschieden Module durchgeführt: DASH, Tkinter , PyQt , wxPython , PyGTK , Canvas…

Diese ist ein wichtiger Bestandteil vieler IT-Projekte, da sie es dem Benutzer ermöglicht, die Funktionalität der Anwendung auf einfache und intuitiv verständliche Weise zu nutzen. In Python gibt es mehrere Module, die es ermöglichen, eine GUI zu erstellen. Eines davon ist Tkinter, das im Standardbibliothek von Python enthalten ist und somit auf jedem System verfügbar ist. Das ist auch der Grund, warum Tkinter in diesem Projekt verwendet wurde.

**Der GUI Programm**

#Import the required Libraries

#import time

import paho.mqtt.client as mqtt

#importierung der Tkinter Bibliothek ,die die Erstellung von grafischen #Benutzeroberflächen ermöglicht.

from tkinter import \*

from tkinter import messagebox

#import PIL Bibliotheken ,die Unterstützung für das Öffnen, Bearbeiten und #Speichern vieler #verschiedener Bilddateiformate bietet.

from PIL import Image,ImageTk

#Declaration von Variablen für die Feuchtigkeit und der Temperature

msg\_temperatur=None

msg\_humidity=None

#Auflistung verschieden Topics

topics = ['fhdo/itp/gp1/12','fhdo/itp/gp1/11','fhdo/itp/gp1/13']

#--------------------------------------------------------------------------#-----------------------------

#----------Funktion , wenn der Broker mit der vorgegebenen Topic eine #Nachricht bekommt-------

def on\_message(client , userdata, message):

msg=str(message.payload.decode())

print(message.topic, '--->', msg)

if message.topic==topics[0]:

if msg=="ANALOG":

Nachricht="Die manuelle Steuerung ist aus!..."

Fact.set(Nachricht)

elif msg=="op":

Fact.set("Welcome....")

elif msg=="wait":

Fact.set("Waiting on Movement....")

if message.topic==topics[1]:

msg\_temperatur="Temperature: "+msg+" °C"

Fact\_temp.set(msg\_temperatur)

if message.topic==topics[2]:

msg\_humidity='Humidity: '+msg+" %"

Fact\_humid.set(msg\_humidity)

#---------------------Funktion beim Anschließen oder Abonnement der Client

def on\_connect (client, userdata, flags, rc):

client.subscribe ('fhdo/itp/gp1/#')

#------------------------Anwendung der MQTT- Protokoll-----------------

client = mqtt.Client()

BROKER\_ADDRESS="broker.emqx.io"

client.on\_connect = on\_connect

client.on\_message =on\_message

client. connect (BROKER\_ADDRESS)

client. loop\_start()

#----Ausführende Funktion , wenn man auf OPEN klickt ----------------

def btn\_on\_click():

client.publish(topics[0], 'open')

Nachricht ="Der Tür ist geöffnet!..."

Fact.set(Nachricht)

#----Ausführende Funktion , wenn man auf CLOSED klickt ----------------

def btn\_off\_click():

client.publish(topics[0], 'off')

Nachricht ="Der Lufter ist aus!..."

Fact.set(Nachricht)

#----Ausführende Funktion , wenn man auf Turn\_On klickt ----------------

def btn\_turn\_click():

client.publish(topics[0], 'turn')

Nachricht ="Der Lufter ist ein!..."

Fact.set(Nachricht)

#----Ausführende Funktion , wenn man auf Turn\_Off klickt ----------------

def btn\_closed\_click():

client.publish(topics[0], 'closed')

Nachricht ="Der Tür ist zu!..."

Fact.set(Nachricht)

#----Ausführende Funktion , wenn man auf MANUELLE klickt ----------------

def btn\_press\_click():

messagebox.askquestion("Manuelle Steurerung",

"Wollen Sie wirklich die manuelle Steuerung aktivieren?")

if 'yes':

client.publish(topics[0], 'press')

Nachricht="Manuelle Steuerung aktiviert!..."

Fact.set(Nachricht)

print(Nachricht)

else:

Nachricht="Die Steuerung ist noch Digital!..."

Fact.set(Nachricht)

print(Nachricht)

#Create an instance of tkinter frame

win = Tk()

win.title("IT-Projekt Gruppe1")

#win.iconbitmap('E:\a.ico')

# Ändert der size des Fenster

win.geometry("1280x620")

# Create text widget and specify size.

titel= Label(win, text="KÜHLRAUM",fg="deep sky blue", bg="royal blue", font="Helvetica 16 bold italic").grid(row=0, column=2, columnspan=2)

Tür=Label(win, text='Tür-Button', fg='black', bg='white', font='verdana').grid(row=3, column=0, columnspan=2)

lufter=Label(win, text='Lufter-Button', fg='black', bg='white', font='verdana').grid(row=3, column=6, columnspan=2)

#glogale Declaration alle Textvariable zum Anzeigen in Labels

global Fact

global Fact\_temp

global Fact\_humid

Fact=StringVar()

Fact\_temp=StringVar()

Fact\_humid=StringVar()

Fact.set("--Stand--")

Fact\_temp.set('--Temperature--')

Fact\_humid.set('--Humidity--')

#Erstellung eine Objekt my\_img zur classe ImageTk

my\_img = ImageTk.PhotoImage(Image.open("fh.png"))

#Erstellung eines Labels zur Darstellung des Bildes

Bild= Label(image=my\_img).grid(row=2, column=2,rowspan=8)

#Erstellung von Labeln zum Anzeigen der Temperatur, der Feuchtigkeit und #des Standes des JOYPIs

temperatur\_info= Label(win, textvariable=Fact\_temp, fg="black", bg="white", font="VERDANA").grid(row=6,column=0,columnspan=2)

feutchtigkeit\_info= Label(win, textvariable=Fact\_humid, fg="black", bg="white", font="VERDANA").grid(row=6,column=6,columnspan=2)

stand= Label(win, textvariable=Fact, fg="black", bg="white", font="VERDANA").grid(row=10,column=2,columnspan=3)

#Erstellung allen gebrauchten Buttons

bouton0=Button(win, text="OPEN", relief=RAISED,fg="SpringGreen4",bg="SpringGreen2", font="VERDANA",padx=30, command= btn\_on\_click).grid(row=4,column=0)

bouton1=Button(win, text="Turn\_ON", relief=RAISED, fg="blue4",bg="turquoise1", font="VERDANA",padx=30, command=btn\_turn\_click ).grid(row=4, column=6)

bouton2=Button(win, text="CLOSED", relief=RAISED, fg="white",bg="grey28", font="VERDANA",padx=30, command=btn\_closed\_click ).grid(row=4,column=1)

bouton3=Button(win, text="Turn\_OFF", relief=RAISED, fg="white",bg="grey28", font="VERDANA",padx=30, command=btn\_off\_click).grid(row=4, column=7)

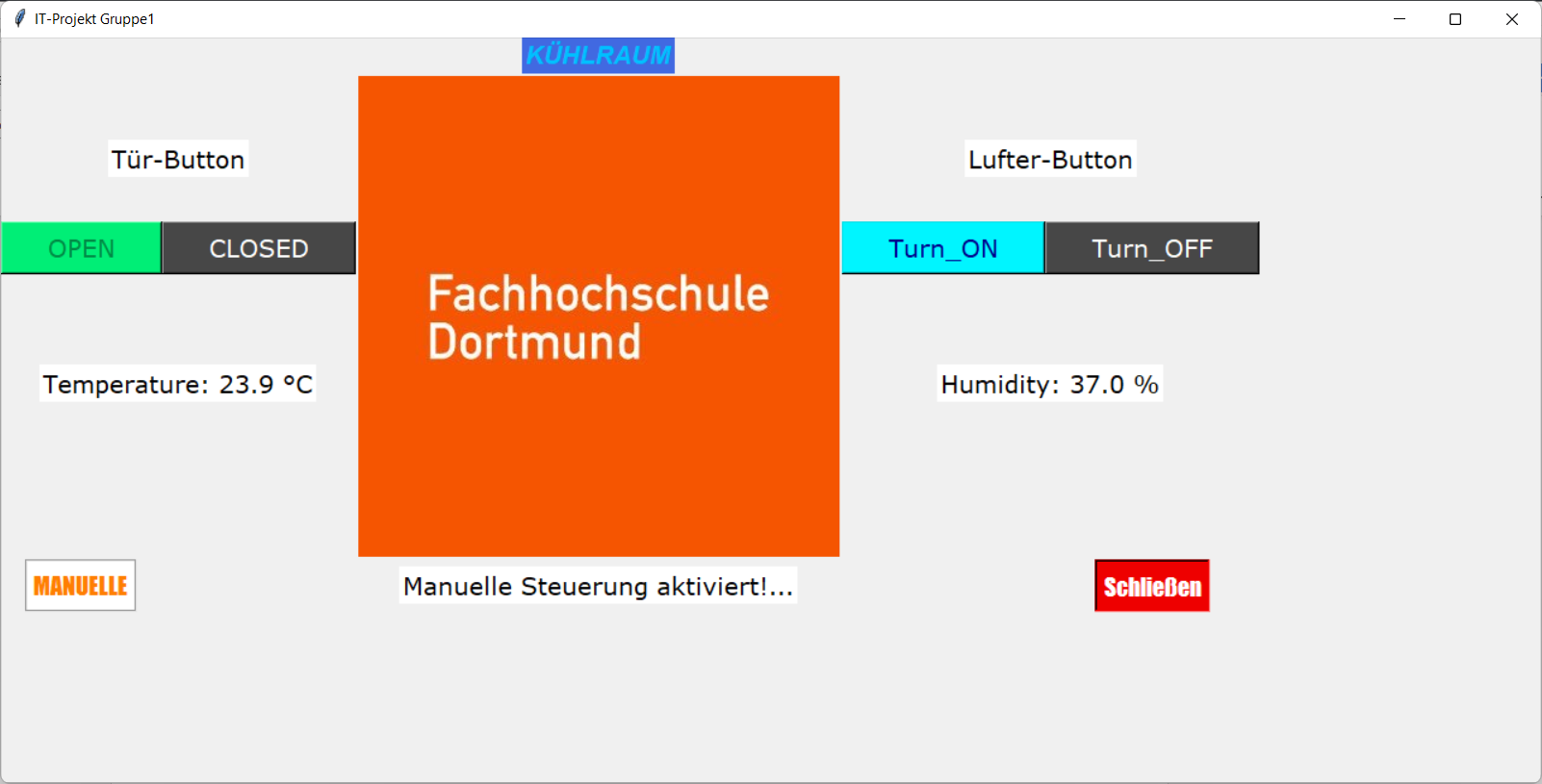
bouton4=Button(win, text="Schließen", relief=SUNKEN, cursor="spider",fg="white",bg="red2", font="IMPACT",command=win.quit).grid(row=10,column=7)

bouton5=Button(win, text="MANUELLE", relief=GROOVE, cursor="spider",fg="DarkOrange1",bg="WHITE", font="IMPACT",command=btn\_press\_click).grid(row=10,column=0)

win.mainloop()

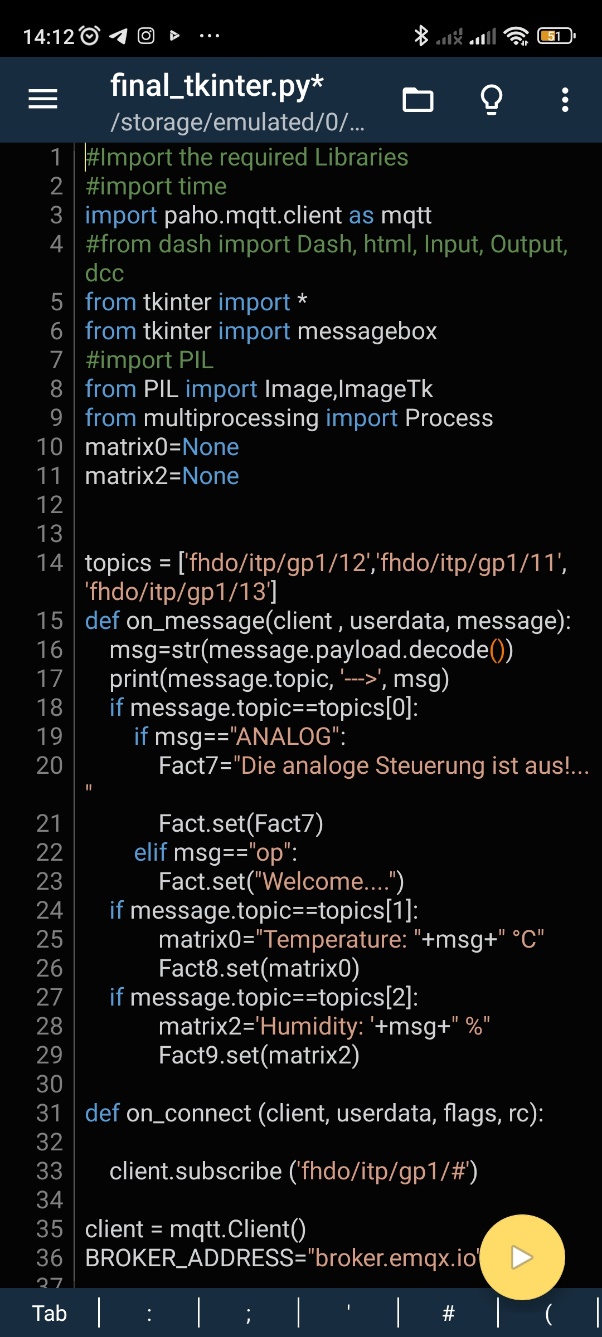
#weist Python an, die Ereignisschleife von Tkinter auszuführen

Die Methode "loop\_start()" startet die Kommunikationsschleife in einem separaten Thread, während die Methode "loop\_forever()" die Programmausführung so lange blockiert, bis die Kommunikationsschleife unterbrochen wird.

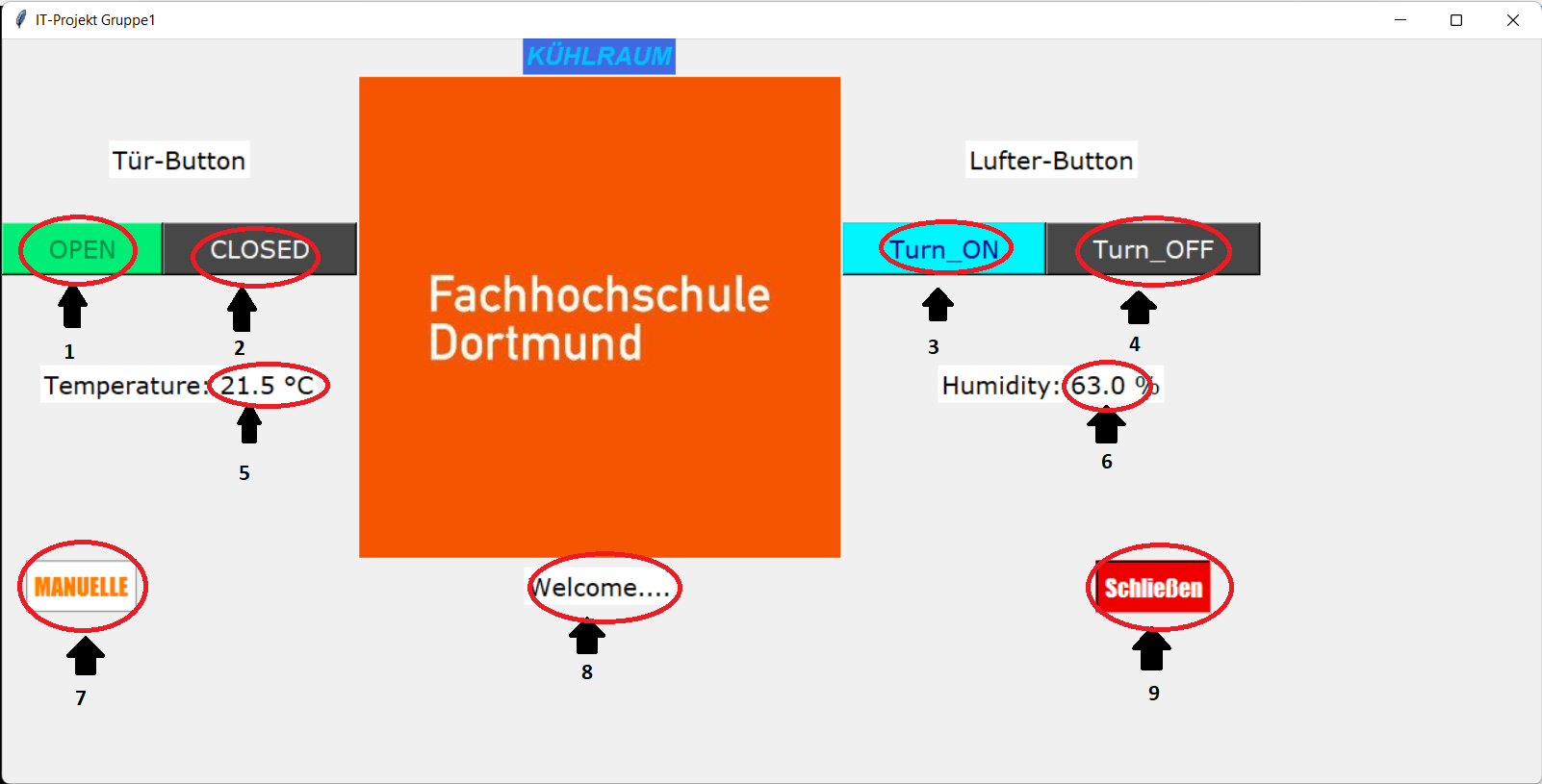


Auf dem Handy:

[Pydroid-Apps für Android](https://play.google.com/store/apps/details?id=ru.iiec.pydroid3&hl=fr&gl=US)

Benutzerdokumentation und Test der Funktionalität

O

1. Open Button
2. Closed Button
3. Turn\_ON Button
4. Turn\_OFF Button
5. Temperatur Anzeige
6. Feuchtigkeit Anzeige
7. Manuelle Button
8. Anzeige von Stände
9. Schließen Button

Youtube-Video:

[Kühlraum IT projekt](https://youtu.be/USVp1kxbuBY)

[Kühlraum IT projekt Updates](https://www.youtube.com/watch?v=Gz9hBCLOhy0)

Zusammenfassung und Ausblick

Zusammenfassend war das Projekt die Steuerungeiner Tür (sowie. Lüfter) eines Kühlraums mit Temperatur- und Feuchtigkeit- Erfassung. Im Laufe des Projekts standen wir vor einer Reihe von Problemen, unter anderem die manuelle und digitale Steuerung können nicht gleichzeitig aktiviert werden, weil diese Beide Funktionen Kontrollflussanweisung (bzw. Ablaufsteuerung oder Flow-Control-Anweisung) sind . Das heißt einer kann nur funktionieren, wenn der andere fertig ist. Um dieses Problem zu beheben, kann man sich einen Thread zunutze machen, damit einer davon unabhängig und gleichzeitig mit anderen Teilen des Programms durchlauft.

Des Weiteren könnte der GUI nicht darauf warten, dass die Funktion des zuletzt gedrückten Buttons vollständig ausgeführt wird, bevor er seinen Status sendet und darstellt, weil der Status nicht von der MQTT Protokoll hängt, u.a. der Statut ist direkt in der Funktion des Buttons .Das kann ein Problem bei mehrfache Druck sein , weil der Python die Nachrichten von den MQTT nach und nach speichert, und es nach reihe verarbeitet .Da der MQTT sofort die Nachrichten schickt , muss eine Wartezeit vor das Veröffentlichen der Nachrichten gesetzt und muss der Statut nicht in der Button-funktion stehen, sondern am Ende der Funktion, die der Button abruft.

Außerdem musste zwei Code geschrieben werden und das liegt daran, dass der Raspberry Pi nicht Tkinter unterstutzt. Deshalb wäre in diesem Fall der Oberfläche mit Dash eine gute Lösung, weil seine Nutzung im Raspberry PI möglich ist.

Es wäre schöner dieses Projekt mit Dash (web design mit Bootstrap, CSS, Dash.DAQ,html5 und JavaScript) zu machen, weil Dash moderner sehen wird. Darüber hinaus wäre es super, wenn sich die Temperatur mit dem Lüfter anschließt, sodass bei höherer Temperatur sich der Lüfter automatisch einschaltet.

Dash bootstrap