Internet de las Cosas Lab Exercises

Clara Gauer

UPV

**Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte BeschreibungLab 7**

Function to measure distance using the HC-SR04 sensorThe sensor is triggered by sending a high pulse of at least 10 microseconds to trigPit the pulse tells the sensor to emit an ultrasonic burstwhen the trigger pin receives the high pulse, the sensor emits ultrasonic sound wavesthe sensor waits for the sound waves to bounce off an object and return to the sensor when the sensor detects the returning echo, the echoPin goes high and stays high until the echo pulse ends

Ein Bild, das Text, Screenshot, Schrift, Dokument enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift, Dokument enthält.

Automatisch generierte Beschreibung

**Lab 9 - GPIO**

Green: GPIO pin is set to ‘OUTPUT’ and is in the ‘HIGH’ state

Red: GPIO pin is set to ‘OUTPUT’ and is set to the ‘LOW’ state

Orange: GPIO pin is set to ‘INPUT’ and the state is ‘HIGH’

Blue: GPIO pin is set to ‘INPUT’ and the state is ‘LOW’

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungGray: GPIO pin is not configured (neither input or output)

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

PWM und LEDs

* Ein Tastgrad von 1 bzw 100% wäre, wie wenn die ganze Zeit obere Spannung anliegen würde
* 0% würde heißen, dass die ganze Zeit keine Spannung anliegt
* Ist in 25% der Zeit die Spannung auf 3.3V und 75% auf 0V, so ergibt sich eine durchschnittliche Spannung von 0,825V

PWM und Servomotoren

* Servos können zwischen 0 und 180 Grad rotieren
* Pulse width
  + A pulse width of 1 ms corresponds to 0 degrees
  + A pulse width of 1.5 ms corresponds to 90 degrees (the midpoint)
  + A pulse width of 2 ms corresponds to 180 degrees
  + These pulses are sent repeatedly at a certain frequency, commonly 50 Hz (50 times per second)
* Duty cycle calculation
  + T =
  + Thus, a pulse width of 1 ms is of the period, corresponding to a duty cycle of 5%
  + Thus, a pulse width of 1.5 ms is of the period, corresponding to a duty cycle of 7.5%
  + Thus, a pulse width of 2 ms is of the period, corresponding to a duty cycle of 10%

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungIn a typical button circuit with a pull-up resistor, the button is connected between the GPIO pin and ground (0V).

Normal state (button not pressed)

* When the button is not pressed, the GPIO pin is connected to the high voltage (3.3V) through the pull-up resistor
* The input reads GPIO.HIGH

Pressed state (button pressed)

* When the button is pressed, it creates a direct connection between the GPIO pin and ground (0V), bypassing the pull-up resistor
* Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

  Automatisch generierte BeschreibungThe GPIO pin is pulled to ground, and the input reads GPIO.LOW

Ein Bild, das Diagramm, Entwurf, Text, Zeichnung enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Schrift, Screenshot, Algebra enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibungtry-except-finally statement

The finally clause always executes whether an exception occurs or not. And it executes after the try clause and any except clase

**Raspberry Pi UART Communication**

The Raspberry PI has two pins that can be used for serial communication. These are known as Tx (transmit) and Rx (receive) pins, and they are located at the GPIO header.

When connected, the TX pin’s role is to send data from the Raspberry Pi to a different device, while the RX pin is used to receive data.

Ein Bild, das Text, Screenshot, Spielstandanzeige, Design enthält.

Automatisch generierte BeschreibungUART utilizes asynchronous serial communication. This means that the data is not sent in a continuous stream. Instead, it is sent in packets. This has the advantages that UART can be used with devices that have different clock speeds.

A packet is a group of bytes that are sent together. The first byte in a packet is the header, which includes information about the rest of the packet, such as how long the data is and which device it’s for. After the header comes the data itself. This can be any number of bytes, including zero bytes. The last byte in a packet is the trailer, which contains information about the rest of the packet, like how many bytes of data there are.

UART is commonly used to connect devices.

When using UART, be sure the baud rates of both devices match. The baud rate is a measure of how much data bits can be sent per second. If the baud rates are different, the devices will not be able to transfer data with each other.

Baud rate: data transmission rate in bits per second. It determines how fast data is sent between devices

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungTimeout: specifies the maximum amount of time, in seconds, that the program will wait for data to be received on the serial port before giving up and returning. This is useful for preventing the program from hanging indefinitely while waiting for input

bytesize: size of each character or byte transmitted, common values are 8

Parity: error-checking mechanism

* N: NONE, no parity bit (most common)
* E: EVEN: parity bit is set to make the number of 1s even
* O: ODD: parity bit is set to make the number of 1s odd

Stop bits

* Used to signal the end of a byte
* Values
  + 1: one stop bit. Most common
  + 1.5: one and a half stop bits

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung .read()

* Reads a specified number of bytes from the serial port
* Size: number of bytes to read, if not specified, the default is 1
* Returns: a bytes object contraining the read data

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung.readline()

* Reads a line of text from the serial port, ending at a newline character \n
* Size (optional): maximum number of bytes to read, including the newline character. If not specified, it will read until a newline is found or the timeout occurs

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

With the KeyboardInterrupt, you can easily exit with Ctrl + C without abruptly having to quit the program.

The ser.write() method has to transfer bytes, so the string must be converted to bytes

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte BeschreibunginWaiting() returns the number of bytes currently in the input buffer. Useful if you want to check if there is data available to read before calling ser.readline()

Ein Bild, das Text, Screenshot, Schrift enthält.

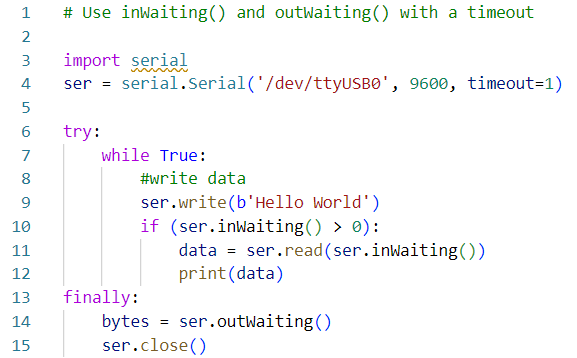
Automatisch generierte BeschreibungEin Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

ser.outWaiting() returns the number of bytes in the outgoing buffer that are waiting to be transmitted. This can be useful to check if all data has been sent before closing the connection

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung



The core function of UART is to transmit and receive serial data. This exercise demonstrates reading data from a serial port, which is a fundamental aspect of UART communication.

Writing and reading data interchangeably: Writing data to the serial port and reading responses simulate bidirectional communication typical in UART

Using inWaiting() and outwaiting(): These methods manage the input and output buffers, crucial for handling data flow in UART communication

**I2C communication**

Ein Bild, das Text, Reihe, Schrift, Diagramm enthält.

Automatisch generierte BeschreibungI2C stands for inter-integrated circuit and is a method designed to allow one chip to talk to another synchronously. It is a hardware protocol which allows multiple slave integrated circuits to communicate with one or more master.

It uses bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock Line (SCL), pulled up with resistors.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

The smbus library is used to communicate with devices over the I2C bus

Bus = smbus.SMBUS(1) creates an instance of the SMBus class, which represents an I2C bus. The 1 specifies the bus number

Read data

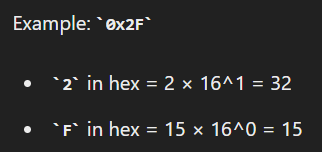
* Read a single byte from the specified register address (0x01) of the device at the specified device address (0x50)
* Device\_adress: I2C address of the device
* Register\_address: address of the register from which to read the byte

Device address

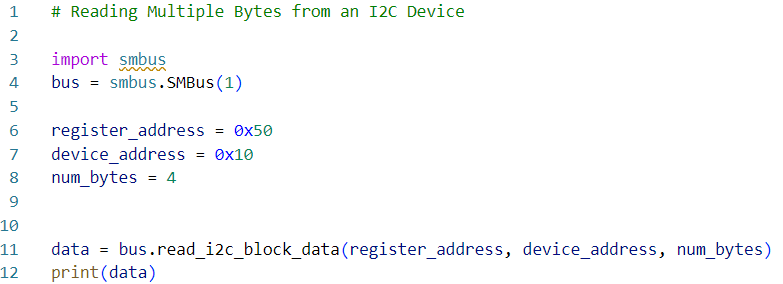
* Device address tells the I2C master which device on the bus should receive the command
* Each I2C device on the bus has a unique address

Register address (accessing specific data)

* Register address tells the device which specific register within its internal memory map should be decreased
* Registers hold specific pieces of data or control bits. For instance, a temperature sensor might have different registers for temperature readings, configuration settings and status flags

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung



Read\_i2c\_block\_data(i2c\_addr, register, length) method is used to read a block of data from an I2C device. It is useful if you need to read multiple bytes from a specific register on the I2C device

* I2c\_addr: the I2C address of the device
* Register: register address from which to start reading data
* Length: number of bytes to read

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungThe device address comes first because it is necessary to specify which device on the I2C bus you are communicating with before any data transactions can occur.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungThe device address is the first parameter because the I2C master needs to specify which device it is addressing on the bus. This address ensures that the subsequent command and data are sent to the correct device. The register address specifies the location within the device’s memory map where the data should be written. After the device is addressed, the master needs to tell the device which register it intends to write to

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung**SPI communication**

Serial Peripheral Interface (SPI) is an asynchronous serial bus commonly used to send data between microcontrollers and small peripherals such as shift registers, sensors and more.

It uses separate clock (SCK) and data lines (MISO, MOSI) along with a select line (SS) to choose among the multiple slave devices.

With SPI, we can interface multiple peripheral devices as SPI slaves and control them with the SPI master. In general, each slave will need a separate SS line. To talk to a particular slave, the SPI master makes that slave’s SS line low and keeps the rest of them high.

One unique benefit of SPI is that data can be transferred without interruption. Any number of bits can be sent in a continuous stream. With I2C and UART, data is sent in packets, limited to a specific number of bits.

Devices communicating via SPI are in a master-slave relationship.

* MOSI (Master Output/ Slave Input): line for the master to send data to the slave
* MISO (master input/ slave output): line for the slave to send data to the master
* SCLK (clock) – line for the clock signal
* Ein Bild, das Text, Schrift, Reihe, Screenshot enthält.

  Automatisch generierte BeschreibungSS (slave select): line for the master to select which slave to send data to

Synchronous: it uses a clock signal to synchronize the data between the master and the slave

Serial: data is transmitted bit by bit along a single wire pair

In SPI, there is only one master in an SPI bus, although multi-master configurations are possible but uncommon.

Maximum number of slaves: in practice, the number of slaves is limited by factors such as signal integrity, wiring complexity and availability of chip select lines

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

SPIDev is a python library used to interface with SPI devices on Linux-based systems like RPi.

Spi = spidev.SpiDev(): creates an instance of the SpiDev class, which represents an SPI interface. This object will be used to configure the SPI settings and perform data transfers

Opening the SPI bus: spi.open(bus, device)

* Opens a connection to an SPI device
* Bus: the SPI bus number
* Device: the SPI device (chip select) on the bus

Selecting the SPI speed: spi.max\_speed\_hz = 100000

* Sets the maximum clock speed for SPI communication. Here it is set to 1 MHz. This determines how fast data is transmitted over the SPI bus

Sending and receiving data: rx\_data = spi.xfer2(tx\_data)

* Spi.xfer2(data): this method performs an SPI transaction. It sends the bytes in ‘tx\_data’ to the SPI device and simultaneously receives bytes from the device
* This method returns a list of bytes received from the SPI device, which is stored in rx\_data
* Xfer2 allows full-duplex communication, meaning data is sent and received simultaneously

Closing the SPI connection: spi.close()

* Closes the SPI connection, freeing any resources used. It’s good practice to close the connection when you’re done using it to avoid conflicts or resource leaks

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibungspi.writebytes(values): write a list of values to the SPI device

|  |  |
| --- | --- |
| Xfer | Xfer2 |
| Transfers a list of values to the SPI device and reads a list of values from the SPI device | Transfers a list of values to the SPI device and reads a list of values from the SPI device |
| May release the SS line between blocks of data wttransfers | Keeps the SS line active for the entire duration of the data transfer |
| More flexible with optional parameters for delay, speed and bits per word |  |

SPI mode

* Clock Polarity (CPOL)
  + CPOL = 0, the clock signal (SCLK) is low when idle
  + CPOL = 1, the clock signal is high when idle
* Clock Phase (CPHA)
  + CPHA = 0, data is sampled on the leading edge of the clock
  + CPHA = 1, data is sampled on the trailing edge of the clock
* Code: spi.mode = 3

Ein Bild, das Text, Schrift, Reihe, Diagramm enthält.

Automatisch generierte Beschreibung

SPI bit order

* To set the bit order, you typically use the spi.lsbfirst attribute
  + Spi.lsbfirst = False: sets the bit order to MSB first
  + Spi.lsbfirst = True: sets the bit order to LSB first

Modify behavior of Chip Select line in SPI communication

* By default, SPI uses an active-low chip select, which means CS line is pulled low to enable communication with the SPI slave device
* Spi.cshigh = false: the chip selects line is active low. This is the default behavior. When communication starts, the CS line is pulled low, and when communication ends, the CS line is pulled high
* Spi.cshigh = True: the chip select line is active high. When communication starts the CS line is pulled high, and when communication ends, the CS line is pulled low

SPI transfer delay

* Configuring a transfer delay can be important to ensure the proper timing between successive data transfers
* This can help in scenarios where the slave device requires some time to process data before the next transfer or to meet specific timing requirements
* Delay\_usecs: parameter which specifies the delay between consecutive SPI transfers in microseconds

Chip select pin

* Spi.no\_cs: True
  + Disables automatic handling of the CS pin by the SPI driver
  + SPI driver will not control the CS pin during SPI transaction

**Lab 10**

Null modem configuration: refers to a method of connecting two devices directly using a serial cable without the need for modems.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

This communication should now take place between two Raspberry Pis.

Hardware connections

* Connect the TX pin of the sender Pi to the RX pin of the receiver Pi
* Connect the RX pin of the sender Pi to the TX pin of the receiver Pi
* Connect GND pin to both raspberry Pis

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungSender program

Receiver program

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Programming of sensors: UART protocol

Ein Bild, das Text, Schrift, Screenshot, Zahl enthält.

Automatisch generierte BeschreibungWhen connecting the Zilog ZMOTION motion sensor, one of the pins indicate if motion has been detected. In serial mode, the one we are going to use is connected via the serial interface and a command is sent to it to find out if there has been a motion detection.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungI2C communication

I2C is a protocol for communication between devices. It transmits data in 8-bit chunks. Many sensors or ADCs produce data that is larger than 8 bits. For example, a 10-bit ADC value needs to be transmitted over I2C.

Original data size: if the original data is larger than 8 bits, it needs to be split into multiple bytes for transmission.

When receiving the data, you need to combine these bytes to reconstruct the original multi-bit value. This ensures you get the correct and precise reading from the sensor.

bus.read\_i2c\_block\_data(SLAVE\_ADDRESS, 0, 2) reads 2 bytes from the I2C device

* Data[0] is the first byte
* Data[1] is the second byte

Combining the bytes

* Bitwise Left Shift (<<): Shifting data[0] left by 8 bits moves its value to the high byte position of a 16-bit integer.
* Bitwise OR (|): The OR operation combines the shifted high byte with the low byte to form the original 16-bit value.

**Lab 11**

MQTT

* Standardbasiertes Messaging-Protokoll, bzw. Reihe von Regeln, die für die Kommunikation zwischen Maschinen verwendet werden
* MQTT wird für Datenübertragung verwendet, da es einfach zu implementieren ist und IoT-Daten effizient übertragen kann

Vorteile von MQTT

* Leicht und effizient
  + Erfordert nur minimale Ressourcen, sodass sie sogar auf kleinen Mikrocontrollern verwendet werden kann
  + MQTT Nachrichtenheader sind klein
* Skalierbar
  + Implementierung erfordert minimale Menge an Code, der im Betrieb wenig Energie braucht
  + Protokoll verfügt außerdem über integrierte Funktionen zur Unterstützung der Kommunikation mit einer großen Anzahl von IoT-Geräten
* Zuverlässig
  + IoT-Geräte verbinden sich über unzuverlässige Mobilfunknetze mit geringer Bandbreite und hoher Latenz
  + Es werden drei Servicequalitätsstufen definiert, um die Zuverlässigkeit für IoT-Anwendungsfälle zu gewährleisten
* Sicher
  + MQTT macht es Erfindern leicht, Nachrichten zu verschlüsseln

MQTT-Komponenten

* MQTT-Client
  + Jedes Gerät, vom Server bis zum Microcontroller, auf dem ein MQTT-Bibliothek läuft
  + Wenn der Client Nachrichten sendet, fungiert er als Herausgeber, und wenn er Nachrichten empfängt, fungiert er als Empfänger
* MQTT-Broker
  + Backend-System, das die Nachrichten zwischen den verschiedenen Clients koordiniert
  + Zu den Aufgaben des Brokers gehören der Empfang und die Filterung von Nachrichten & Identifizierung von Kunden
* MQTT-Verbindung
  + Client und Broker beginnen ihre Kommunikation über eine MQTT-Verbindung.
  + Clients initiieren die Verbindung indem sie eine CONNECT-Nachricht an den MQTT-Broker senden

Wie funktioniert MQTT?

1. Ein MQTT-Client stellt eine Verbindung mit dem MQTT-Broker her
2. Sobald die Verbindung hergestellt ist, kann der Client entweder Nachrichten veröffentlichen, bestimmte Nachrichten abonnieren oder beides tun
3. Wenn der MQTT-Broker eine Nachricht erhält, leitet er sie an interessierte Abonnenten weiter

* Ein Bild, das Text, Screenshot, Schrift enthält.

  Automatisch generierte Beschreibungcallback function: Triggered when trying to connect to the MQTT broker
* client: is the client instance connected this time
* userdata: is users' information, usually empty. If it is needed, you can set it through user\_data\_set function.
* flags: save the dictionary of broker response flag.
* rc: response code. Pay attention mainly to whether the response code is 0 or not.

Explanation

* on\_connect(client, userdata, flags, rc)
  + defines a callback function on\_connect that is called when the client connects to the MQTT broker
  + inside the function, it checks if the connection was successful (rc == 0). If successful, it prints “Connected success”, otherwise it prints an error message with the error code
* client = mqtt.Client(): creates a new instance of the Client class
* client.on\_connect = on\_connect: this sets on\_connect callback to the previously defined on\_connect function
* client.connect("broker.emqx.io", 1883, 60): this line initiates a connection to the MQTT broker
  + broker.emqx.io: address of the broker
  + 1883: port it is connected to
  + 60: keepalive interval
* client.loop\_forever()
  + starts a network loop that runs forever, processing network traffic and dispatching callbacks. This call is blocking, which means the program will stay in this loop, handling all MQTT network traffic and callbacks, until the program is terminated

Ein Bild, das Text, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

* Ein Bild, das Text, Screenshot, Schrift enthält.

  Automatisch generierte Beschreibungon\_connect(client, userdata, flags, rc)
  + This function is called when the client successfully connects to the MQTT broker
  + Parameters
    - Client: the client instance that is calling the callback
    - Userdata: user-defined data of any type that is passed as the third argument when creating a new client instance
    - Flags: response flag sent by the broker
    - Rc: connection result code. A value of 0 means the connection was successful
  + Functionality
    - it prints the topic and payload of the received message. This allows you to see what data has been published to the topic
* client = mqtt.Client()
  + creates a new MQTT instance
* client.on\_connect = on\_connect
  + sets the on\_connect callback function
* client.on\_message = on\_message
  + Sets the on\_message callback function
* client.will\_set('raspberry/status', b'{"status": "Off"}')
  + sets the last will message. This message is published by the broker if the client disconnects unexpectedly. (power failure/ network issue)
* client.connect("broker.emqx.io", 1883, 60)
  + purpose: establishes a connection to the MQTT broker
  + parameter
    - broker.emqx.io: address of the MQTT broker
    - 1883: port to connect to on the broker
    - 60: keepalive interval in seconds. The maximum time that should pass without communication between the client and the broker
* Client.loop\_forever()
  + Purpose: starts a blocking network loop that processes network traffic, dispatches callbacks and handles reconnecting
  + Functionality: the loop will run indefinitely, ensuring that the client can process incoming messages and handle connections automatically

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungEjemplo básico de publicación usando el servidor MQTT mosquitto y la librería paho. Genera valores aleatorios como si fuesen los valores obtenidos de un sensor y los publica cada 5s

* On\_connect
  + Function is called when the client successfully connects to the MQTT broker
  + It prints the broker address, port, flags and connection result code
* On\_publish callback
  + This function is called when a message is published
  + It prints a confirmation message with the message ID (mid) of the published message

Creating and configuring MQTT client

* client = mqtt.Client(client\_id=CLIENT\_ID, clean\_session=True,

userdata=None, protocol=mqtt.MQTTv311, transport="tcp")

* + client\_id = CLIENT\_ID: client id for the MQTT client
  + clean\_session = True: if true, the broker will clean all messages and subscriptions on disconnect
  + userdata=None: user-defined data to pass to callbacks
  + protocol=mqtt.MQTTv311: the MQTT protocol version to use
  + transport = tcp: transport protocol to use
* client.on\_connect = on\_connect
  + sets the on\_connect callback function
* client.on\_publish = on\_publish
  + sets the on\_publish callback function

Set username and password

* client.username\_pw\_set(None, password=None)
  + purpose: sets the username and password for the MQTT client

Connecting to the broker

* client.connect(THE\_BROKER, port=1883, keepalive=60)
* client.loop\_start()
  + purpose: establishes a connection to the MQTT broker and starts the network loop
  + parameters
    - THE\_BROKER: address of the MQTT broker
    - Port= 1883: port to connect to on the broker
    - Keepalive=60: keepalive interval in seconds
  + Functionality
    - Client.connect(): connects to the broker
    - Client.loop\_start(): starts a new thread to handle network traffic, making the client non-blocking

Publish messages in a loop

* Client.publish()
  + Publishes the generated random number to the specified topic

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungConstants: THE\_TOPIC is the topic you subscribe to

Callback functions

* On\_connect
  + Client.subscribe(THE\_TOPIC; qos=0): Subscribes to the topic defined in THE\_TOPIC with QoS level 0
* On\_message: prints the topic and payload of the received message

Creating and configuring the MQTT clients

* client = mqtt.Client(client\_id=CLIENT\_ID, clean\_session=True, userdata=None, protocol=mqtt.MQTTv311, transport="tcp")
* client.on\_connect = on\_connect
* client.on\_message = on\_message

Setting username and password

* client.username\_pw\_set(None, password=None)
  + sets the username and password for the MQTT client. In this case, both are set to none

Connecting to the broker

* client.connect(THE\_BROKER, port=1883, keepalive=60)
  + establishes a connection to the MQTT broker
  + THE\_BROKER: address of the MQTT broker
  + Port=1883: port to connect on the broker
  + Keepalive=60: keepalive interval in seconds

This script will continuously run, listening for system-related messages from the MQTT broker and printing any received messages to the console.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

* Def settimeout(duration):
  + A placeholder function, which typically would set a timeout for a connection. Here, it does nothing (pass)

Wifi connection and MQTT client setup

* ufun.connect\_to\_wifi(wifi\_ssid, wifi\_passwd)
  + connects to the specified WiFi network using the provided SSID and password
* client = MQTTClient(dev\_id, broker\_addr, 1883, user=user\_name, password='None')
  + creates an instance of the MQTT client with the given device ID, broker address, port (1883), username and password

Connecting to the broker with error handling

* try-except-block: attempts to connect to the MQTT broker. If it fails (raises an OSError) it prints and error message and exits the program using sys.exit()

Publish sensor data in a loop

* creates an infinite loop to continuously read sensor data and publish it
* client.publish: publishes the sensor\_value dictionary as a JSON string to the specified MQTT topic