

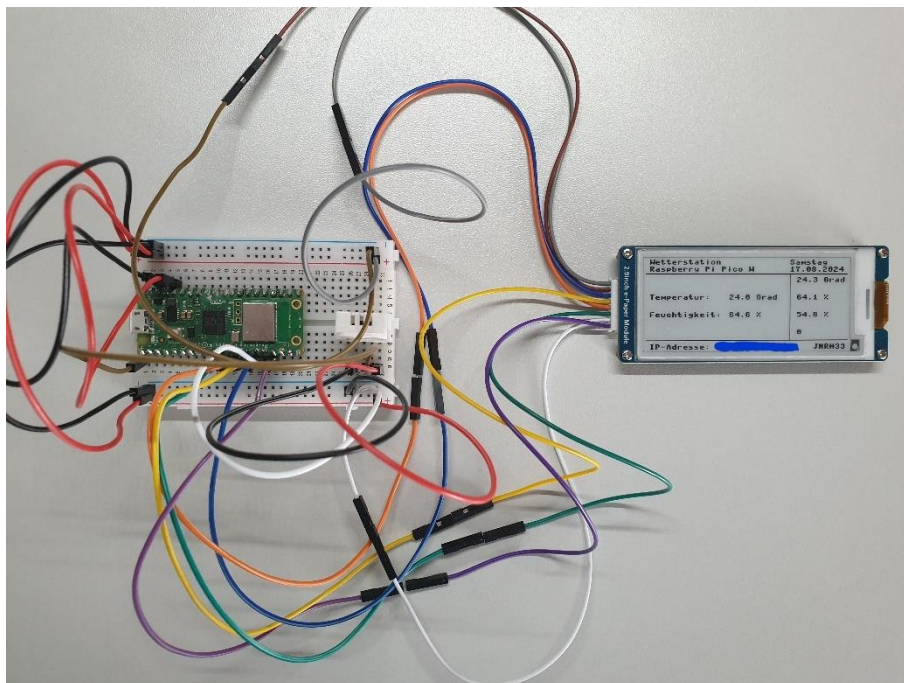
Projekt – Heterogenous Computing

Remote-Wetterstation – Aufbau

In diesem Kapitel geht es um den Aufbau der Remote-Wetterstation. Die Remote-Wetterstation besteht aus zwei Bestandteilen: der Innen- und Außenstation. Hierbei wird zuerst der Aufbau der Innenstation beschrieben. Danach geht es um die Webseite dieser Station. Im Anschluss wird der Aufbau der Außenstation vorgestellt. Dabei hat diese auch eine eigene Webseite, die darauffolgend beschrieben wird. Zum Schluss wird noch der Einsatz von NFC-Sticker erklärt.

Aufbau der zentralen Station

Im Folgenden wird der Aufbau der Remote-Wetterstation veranschaulicht:



In der obigen Abbildung befindet sich der Aufbau des ersten Teils der Remote-Wetterstation: die Innenstation. Sie besteht aus den folgenden Komponenten:

- Raspberry Pi Pico W¹
- DHT22 (Temperatur- und Feuchtigkeitssensor)²
- E-Paper Display³

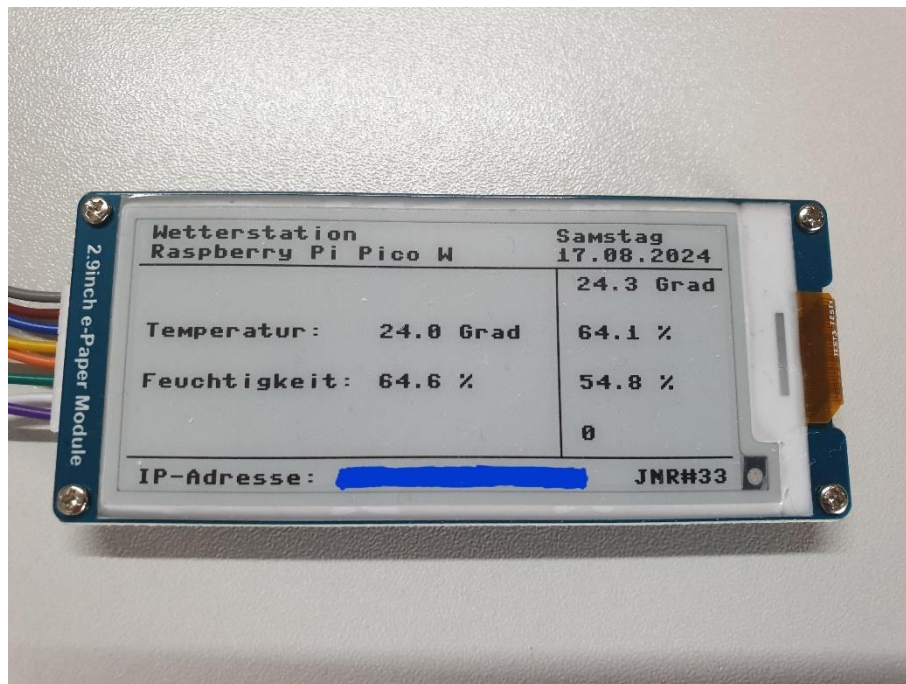
¹ Daten zum Mikrocontroller befinden sich auf der folgenden Seite:

<https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html#picow-technical-specification> (Letzter Zugriff am 13.08.2024).

² Die Daten zu DHT22 befinden sich auf <https://www.berrybase.de/dht22-digitaler-temperatur-und-luftfeuchtesensor> (Letzter Zugriff am 13.08.2024).

³ Die Daten zum Display befinden sich auf <https://www.waveshare.com/2.9inch-e-paper-module.htm> (Letzter Zugriff am 13.08.2024).

Sie ist das Zentrum der Remote-Wetterstation. Hier werden zum einen die Daten des eigenen Temperatur- und Feuchtigkeitssensors gelesen und zum anderen auch die Daten der Sensoren der Außenstation angefragt. Alle Daten werden auf das E-Paper Display angezeigt.



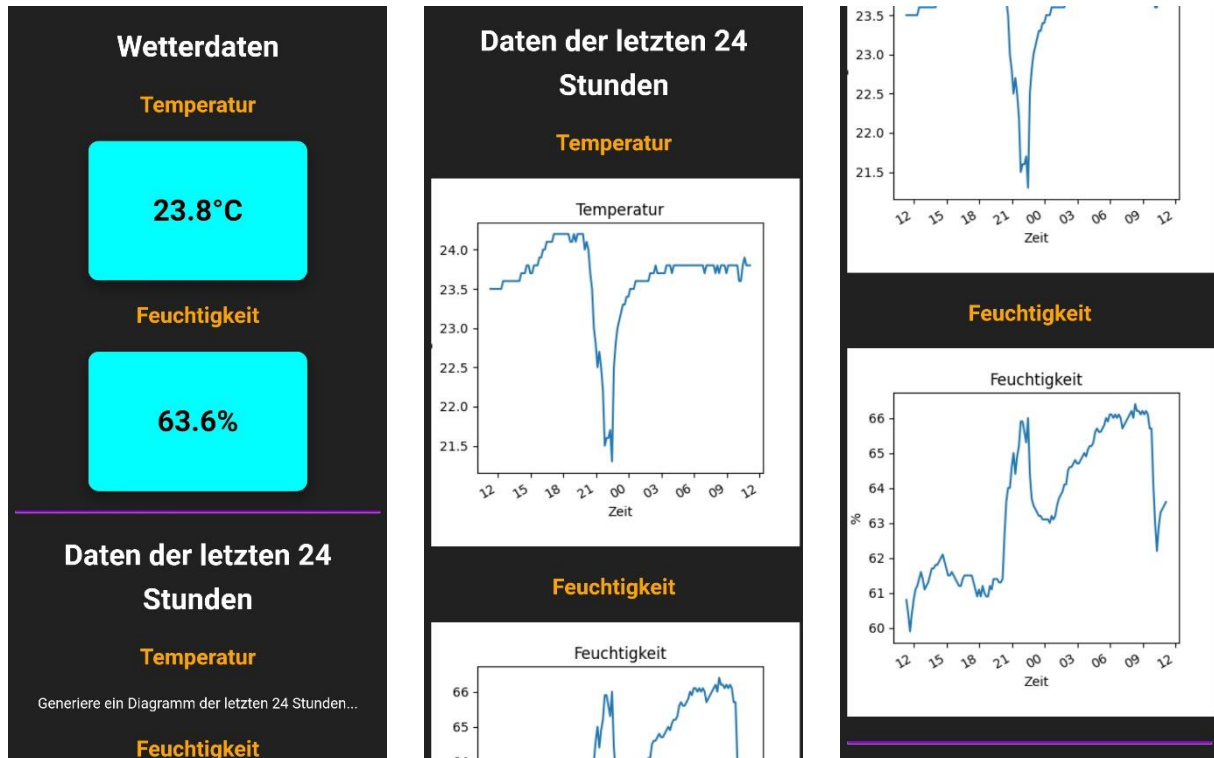
In der obigen Abbildung sieht man die Informationen, die das E-Paper Display anzeigt:

- Titel
- Wochentag mit Datum
- Daten des Temperatur- und Feuchtigkeitssensors (DHT22)
- Daten der Außenstation in der folgenden Reihenfolge:
 - Temperatur
 - Feuchtigkeit
 - Lichtintensität
 - Erkennung von Regen
- IP-Adresse dieser Station
- Meine Initialen

Für dieses Projekt habe ich mich für ein E-Paper Display entschieden, da die Technologie sehr energiesparsam ist und die Station soll durch die fehlende Lichtquelle nicht stören. Hierbei muss man angeben, dass dieses Display keine Hintergrundbeleuchtung besitzt. Das bedeutet, dass man im Dunkeln die Daten der Sensoren nicht lesen kann. Aus diesem Grund habe ich mich auch für einen Raspberry Pi Pico W entschieden. Dieser soll nämlich den Nachteil des Displays mithilfe eines Webserver kompensieren.

Aufbau der Webseite der Innenstation

Die Webseite der Innenstation wurde mithilfe von HTML entwickelt. Hierbei werden die Daten der gemessenen Sensoren angezeigt. Zudem werden auch die Daten der letzten 24 mithilfe zweier Graphen dargestellt.

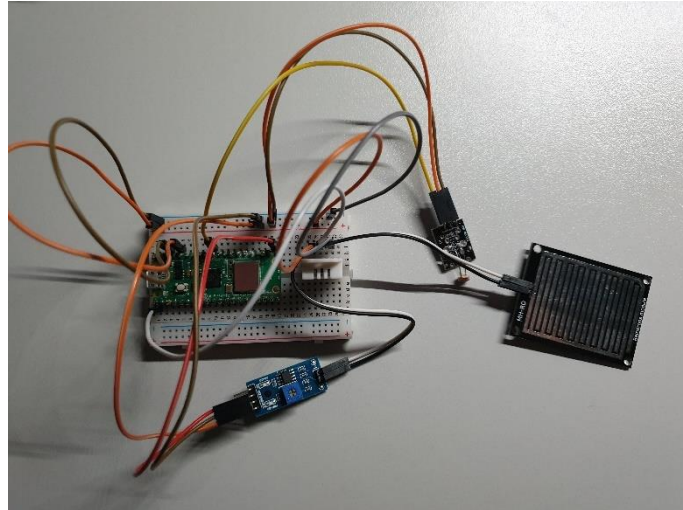


In den obigen drei Abbildungen wird die Webseite der Innenstation dargestellt. Wenn man über die IP-Adresse die Webseite aufruft, werden die gemessenen Daten zur Temperatur und Feuchtigkeit angezeigt. Eine Besonderheit dieser Webseite ist, dass auch die Daten der letzten 24 Stunden dargestellt werden. Mithilfe des Frameworks **PyScript**⁴ kann man Python-Skripte im Browser ausführen lassen. Hierbei konnte ich mit der Bibliothek **Matplotlib** Graphen basierend auf diese Daten generieren und anzeigen lassen. Jedoch werden diese Diagramme erst nach einer gewissen Ladezeit erstellt. Zur Veranschaulichung dieser Zeit habe ich im gleichen Repository auf GitHub das Video „Ladezeiten_Webseite_mit_PyScript.mp4“ hochgeladen. In diesem Video wurde die Webseite auf einem Handy aufgerufen.

Aufbau der Außenstation

Neben der zentralen Station gibt es noch eine Außenstation, die aus weiteren Sensoren besteht und draußen angebracht werden soll. Dabei habe ich mich gegen ein Display entschieden, da die Daten auch über HTTP angefragt werden können.

⁴ Weitere Informationen zum Framework befinden sich auf der folgenden Webseite: <https://pyscript.net/> (Letzter Zugriff am 13.08.2024).



Die Außenstation besteht aus den folgenden Komponenten:

- Raspberry Pi Pico W¹
- Temperatur- und Feuchtigkeitssensor (DHT22)²
- Fotowiderstand (KY-018)⁵
- Regentropfen Sensor Modul⁶

Diese Station misst alle zehn Minuten Daten von den Sensoren, die dann von der Innenstation über GET-Requests angefragt werden. Außerdem wurden zwei neue Sensoren hinzugefügt. Zum einen wird ein Fotowiderstand verwendet. Zum anderen wird ein Regentropfen Sensor Modul genutzt. Wenn Licht auf den Fotowiderstand fällt, dann ändert sich der Widerstand des Sensors. Dieser Wert kann gemessen werden, um die Helligkeit des Lichts zu bestimmen.⁵ Das Modul für den Regen erkennt nur, ob es regnet. Hierbei kann nicht gemessen werden, wie viel Liter pro m² Regen fallen.

⁵ Die Daten zum Sensor werden auf der folgenden Webseite beschrieben: <https://www.az-delivery.de/products/licht-sensor-modul> (Letzter Zugriff am 13.08.2024).

⁶ Die Daten zum letzten Sensor werden auf <https://www.az-delivery.de/products/regen-sensor-modul> (Letzter Zugriff am 13.08.2024) beschrieben.

Aufbau der Webseite der Außenstation

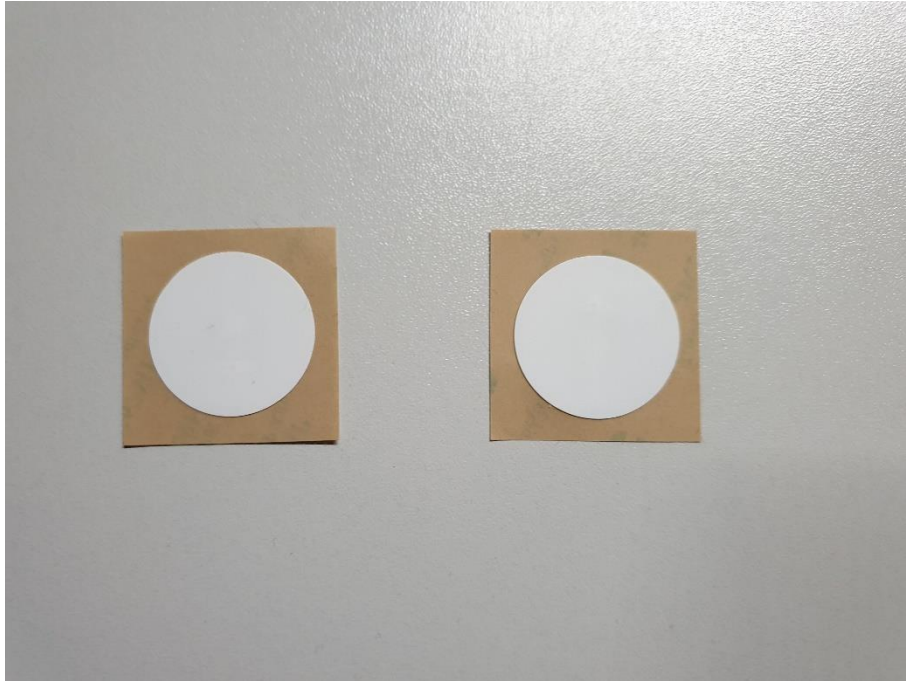
Des Weiteren stellt die Außenstation auch einen Webserver zur Verfügung, damit die Daten der Sensoren auch im Browser gelesen werden können. Hierbei wurde die Webseite aber sehr einfach gehalten, denn diese zeigt nur die letzten gemessenen Daten an:



Die zweite Website hat auch das gleiche Design wie die Website der Innenstation, um die Konsistenz zwischen beiden zu gewährleisten. Des Weiteren wird der Messwert des Fotowiderstands in die Intensität des Lichts in Prozent umgerechnet. Für die Intensität wird nun der Messwert durch 2^{16} (der maximale Widerstandswert des Sensors) dividiert und dieses Ergebnis wird dann mit 100 multipliziert. Für die Erkennung von Regen wird nur 0 oder 1 angegeben.

Zugang zu den Webseiten vereinfachen

Die Remote-Wetterstation hat aber noch ein weiteres Problem: Die beiden Webseiten der Stationen können nur über eine statische IP-Adresse aufgerufen werden. Damit alle Personen im gleichen Haushalt, ohne sich die Adressen merken zu müssen darauf zugreifen können, habe ich zusätzlich NFC-Sticker verwendet.



In der Abbildung oben sind NFC-Sticker zu sehen, die verwendet werden können, um anderen Personen den Zugang zu den Webseiten zu ermöglichen. Der linke Sticker enthält die Adresse der Zentrale, während der rechte Sticker die IP-Adresse der Außenstation enthält.

Übersicht der Funktionen

In diesem Kapitel wird der Quellcode und die Funktionsweise der Remote-Wetterstation beschrieben. Hierbei wird zuerst der Programmcode der Innenstation betrachtet. Danach geht es um den Code für die Webseite dieser Station, da hier das Framework **PyScript**⁴ verwendet wird. Zum Schluss geht es um den Quellcode der Außenstation.

Quellcode der Innenstation

Der Quellcode wird nun in den folgenden Codeabschnitten beschrieben und erklärt. Hierbei werden nur die wichtigen Teile des Programmcodes betrachtet.

```
def main():  
    initialize_led()  
    initialize_dht22()  
    setup_display()  
    [...]
```

In der Hauptfunktion des Programms werden zuerst die eingebaute LED und der Sensor (DHT22) initialisiert. Darauf folgend wird das E-Paper Display konfiguriert. Hierbei werden alle statischen Elemente bereits auf das Display geschrieben, da diese sich niemals ändern.

Durch die Konfiguration werden diverse Elemente wie der Titel und die horizontalen sowie vertikalen Linien hinzugefügt. Zudem werden auch die Beschreibungen „Temperatur: “ und „Feuchtigkeit: “ auf das Display geschrieben. Des Weiteren werden auch alle Vorkommen von „Grad“ und „%“ eingefügt. Die letzten beiden Elemente, die noch in der Funktion *setup_display()* hinzugefügt werden, sind „IP-Adresse: “ und meine Initialen.

```
try:
    connection, ip = connect_to_wifi()
except Exception as e:
    [...]

add_IP_to_display(ip)
# date
set_date_time_NTP()
# Initialize of variables
UTC_OFFSET = -1
last_temp = -1
last_humi = -1
last_weekday_number = -1
last_temp_outdoor = -1
last_humi_outdoor = -1
last_rain_outdoor = -1
last_light_outdoor = -1
first_digit_of_minute = -1
```

Nach der Konfiguration des E-Paper Displays wird eine Internetverbindung aufgebaut. Hierbei wird die statische IP-Adresse auf das Display geschrieben. In *connect_to_wifi()* wird in einer while-Schleife wiederholt versucht, eine Verbindung herzustellen. Zudem wird auch der Energiesparmodus des WLAN-Chips ausgeschaltet, da der Pico als Server dienen soll. Nachdem die IP-Adresse auf das Display hinzugefügt wurde, wird mithilfe von NTP (Network Time Protocol) das Datum und die Uhrzeit (Zeitzone: UTC) initialisiert. Anschließend werden diverse notwendige Variablen für die Hauptschleife initialisiert.

```
while True:

    temp_first_digit_minute = get_first_digit_of_minute()
    if (first_digit_of_minute != temp_first_digit_minute):
        first_digit_of_minute = temp_first_digit_minute

    try:
        UTC_OFFSET = calculate_utc_offset(time.localtime())
        last_weekday_number = update_date_values(UTC_OFFSET,
last_weekday_number)

        last_temp, last_humi, last_temp_outdoor, last_humi_outdoor,
last_rain_outdoor, last_light_outdoor = update_measure_values(last_temp,
last_humi, last_temp_outdoor, last_humi_outdoor, last_rain_outdoor,
last_light_outdoor)

    except Exception as e:
        [...]

    time.sleep(0.1)
    run_server(connection)
```

Im obigen Codeabschnitt befindet sich die Hauptschleife der Innenstation. Alle zehn Minuten sollen Daten des Sensors gelesen werden. Dafür wird überprüft, ob sich die erste Ziffer der Minute geändert hat. Wenn dieser Fall eintritt, dann wird zuerst der Versatz zwischen UTC und unserer Zeitzone berechnet. Wegen der Sommer- und Winterzeit ist dieser Versatz nicht konstant.


```
def update_date_values(UTC_OFFSET, last_weekday_number):  
    date = time.localtime(time.time() + UTC_OFFSET)  
    weekday_number = date[6]  
    if last_weekday_number != weekday_number:  
        update_date_on_display(weekday_number, date)  
        last_weekday_number = weekday_number  
  
    return last_weekday_number
```

Danach wird getestet, ob sich das gespeicherte Datum geändert hat. Bei einer Änderung des Wochentages wird das Datum und der neue Wochentag auf das E-Paper Display geschrieben.

Nach der Überprüfung einer möglichen Änderung des Wochentages werden neue Werte von den Sensoren gemessen oder über HTTP angefordert. Dabei werden diese Daten nur aktualisiert, falls eine Änderung stattgefunden hat.

```
def update_measure_values(last_temp, last_humi, last_temp_outdoor,  
last_humi_outdoor, last_rain_outdoor, last_light_outdoor):  
    [...]  
    # indoor  
    temp_value, humi_value = measure_dht22()  
    temp_queue.pop(0)  
    temp_queue.append(temp_value)  
    humi_queue.pop(0)  
    humi_queue.append(humi_value)  
  
    # outdoor  
    temp_outdoor_value = get_outdoor_sensor_value('/temp_value')  
    humi_outdoor_value = get_outdoor_sensor_value('/humi_value')  
    rain_outdoor_value = get_outdoor_sensor_value('/rain_value')  
    light_outdoor_value = get_outdoor_sensor_value('/light_value')  
    [...]
```

Zuerst werden neue Daten des angeschlossenen Sensors (DHT22) gelesen und in zwei verschiedene Queues gespeichert. Diese Queues werden benötigt, um einen Graphen mithilfe des Frameworks **PyScript** zu erstellen. Dabei ist die Größe der gewählten Datenstruktur wegen des zehn Minuten Takts auf $(60/10) * 24$ beschränkt, damit die Graphen die Werte der letzten 24 Stunden anzeigen kann. Im Anschluss werden die Daten zu Temperatur, Feuchtigkeit, Lichtintensität sowie Erkennung von Regen angefragt und in Variablen gespeichert.

```
change = False

if last_temp != temp_value:
    set_value_to_buffer(temp_value, 120, 50)
    change = True
    last_temp = temp_value

[...]

if last_temp_outdoor != temp_outdoor_value:
    set_value_to_buffer(temp_outdoor_value, 220, 25)
    change = True
    last_temp_outdoor = temp_outdoor_value

[...]

if change:
    get_led().on()
    update_display()
    change = False
    get_led().off()

return last_temp, last_humi, last_temp_outdoor, last_humi_outdoor,
last_rain_outdoor, last_light_outdoor
```

Für alle sechs Messwerte wird überprüft, ob eine Änderung stattgefunden hat. Wenn eine Änderung stattgefunden hat, dann wird der neue Wert in den Puffer des E-Displays geschrieben und zwischengespeichert. Hierbei soll das Display nur einmal aktualisiert werden.

Zusätzlich leuchtet noch die eingebaute LED, wenn Daten ersetzt werden. In der Funktion *update_measure_values* werden alle Messwerte zur Speicherung zurückgegeben.

In der Hauptschleife des Programmcodes soll auch in jeder Iteration die Funktion *run_server* aufgerufen werden:

```
def run_server(connection):  
    [...]  
  
    try:  
        ready_to_read, _, _ = select.select([connection], [], [], 1)  
        if ready_to_read:  
            client, _ = connection.accept()  
            request = client.recv(1024)  
  
            html = webpage_indoor(temp_value, humi_value, temp_queue,  
hum_i_queue)  
            response = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n"  
+ html  
            client.send(response)  
            client.close()  
        except Exception as e:  
            [...]
```

In der obigen Funktion wird zuerst überprüft, ob es eine Anfrage von einem Client gibt. Wenn die Webseite angefragt wird, dann wird diese mithilfe der benötigten Daten aufgebaut und wieder über HTTP zurückgegeben.

Wenn eine Exception im Programm der Innenstation geworfen und gefangen wird, dann wird die Funktion *mark_exception_on_display* aufgerufen. In dieser Funktion wird ein Dreieck mit einem Ausrufezeichen auf das Display geschrieben. Dieses Symbol soll dabei helfen Fehler im Programm zu erkennen, wenn der Raspberry Pi Pico W nicht an einem Computer angeschlossen wurde.



In der obigen Abbildung sieht man auf der rechten Seite des E-Paper Display das Symbol, wenn ein Fehler aufgetreten ist. Zusätzlich erkennt man im Hintergrund auch noch den vorherigen Zustand des Displays.

HTML-Code der zugehörigen Webseite

Die Webseite besteht aus drei wichtigen Bestandteilen: Die Einbindung von **PyScript**⁴, die Boxen mit den Wetterdaten sowie den Python-Code für die Diagramme. Die Verwendung von **PyScript**⁴ erfolgt ohne Installation des Frameworks. Hierfür muss man nur die folgenden zwei Zeilen in den Header einfügen:

```
<link rel="stylesheet"
href="https://pyscript.net/latest/pyscript.css">
<script defer
src="https://pyscript.net/latest/pyscript.js"></script>
```

Der Nachteil von **PyScript**⁴ ist, dass die Webseite mit dem Framework bei jedem Aufruf auf jedem Gerät geladen werden muss. Die Ladezeiten werden im Video „Ladezeiten_Webseite_mit_PyScript.mp4“ veranschaulicht.

```
<h2 style="
    text-align: center;
    margin-top: 5%;
    margin-bottom: 5%;
    color: orange">Temperatur</h2>
<div style="
    border: 1px solid #000000;
    background-color: aqua;
    padding: 5%;
    margin: 20px auto;
    width: 50%;
    text-align: center;
    border-radius: 12px;
    box-shadow: 0px 8px 16px rgba(0, 0, 0, 0.3);
">
    <p style="font-size:200%"><b>{temp_value}°C</b></p>
</div>
```

Im obigen Abschnitt befindet sich die Definition der Untertitel und die blauen Boxen mit den Messwerten. Diese Werte werden durch die Formatierung des Strings eingesetzt.

```
<py-script>

import micropip
import asyncio

from datetime import datetime, timedelta

async def main():
    await micropip.install('matplotlib')
    import matplotlib.pyplot as plt
    from matplotlib.dates import DateFormatter

    # x-Axis for the last 24 hours
    now = datetime.now()
    time_labels = [now - timedelta(minutes=i*10) for i in
range((60//10) * 24)]
    time_labels.reverse()

    def create_plot(queue, title, ylabel):
        fig, ax = plt.subplots(figsize=(4, 4))
        ax.plot(time_labels, queue)
        ax.set(title=title, xlabel='Zeit', ylabel=ylabel)
        ax.xaxis.set_major_formatter(DateFormatter('%H'))
        fig.autofmt_xdate()
        return fig

    # Canvas-Element anpassen
    Element('output_temp').write(create_plot({temp_queue}, 'Temp
eratur', '°C'))
    Element('output_humi').write(create_plot({humi_queue},
'Feuchtigkeit', '%'))

    asyncio.ensure_future(main())

</py-script>
```


Im letzten gezeigten Abschnitt des HTML-Codes befindet sich der Python-Code im *py-script* Tag. Durch *asyncio.ensure_future(main())* wird sichergestellt, dass *main* als asynchrone Funktion aufgerufen wird. Hierbei soll zuerst **Matplotlib** installiert werden, bevor es in *create_plot* genutzt wird. In dieser Hauptfunktion des Python-Skripts werden die einzelnen beiden Diagramme zu Temperatur und Feuchtigkeit erstellt und in die div-Elemente „output_temp“ und „output_humi“ eingesetzt.

Quellcode der Außenstation

Der Programmcode der Außenstation ist nahezu identisch zum Quellcode der Innenstation. Hierbei werden nur die Unterschiede zwischen beiden Programmen beschrieben und erklärt.

```
def main():  
    [...] # Initialisierung der LED und Sensoren  
  
    led = get_led()  
    led.on()  
  
    try:  
        connection, ip = connect_to_wifi()  
    except Exception as e:  
        machine.reset()  
  
    led.off()  
    [...]
```

Auch wie beim Programmcode der vorherigen Station werden hier als Erstes die eingebaute LED sowie die Sensoren initialisiert. Im Anschluss wird eine Verbindung zum Internet aufgebaut. Da kein Display vorhanden ist, leuchtet die LED so lange bis eine Verbindung etabliert werden konnte. Dadurch kann man eventuelle Verbindungsprobleme besser erkennen, wenn der Raspberry Pi Pico W am Strom angeschlossen wurde.

```
first_digit_of_minute = -1
while True:

    temp_first_digit_minute = get_first_digit_of_minute()
    if (first_digit_of_minute != temp_first_digit_minute):
        first_digit_of_minute = temp_first_digit_minute

    try:
        temp_value, humi_value = measure_dht22()
        light_value = measure_photo_resistor()
        rain_value = measure_raindrop()

    except Exception as e:
        machine.reset()

    time.sleep(0.1)
    run_server(connection)
```

In der Hauptschleife des Programms werden neue Daten von den Sensoren alle zehn Minuten gemessen und gespeichert. Dabei wird auch jede 0,1 Sekunde die Funktion *run_server* aufgerufen. Durch das fehlende Display existieren in der *main.py* im Verzeichnis *outdoor* keine Überprüfungen auf Änderungen der Messwerte.

```
def run_server(connection):  
    [...]  
    try:  
        ready_to_read, _, _ = select.select([connection], [], [], 1)  
        if ready_to_read:  
            led = get_led()  
            led.on()  
            [...]  
            if '/temp_value' in request:  
                response = f"HTTP/1.1 200 OK\nContent-Type:  
text/plain\n\n{temp_value}"  
            elif '/humi_value' in request:  
                response = f"HTTP/1.1 200 OK\nContent-Type:  
text/plain\n\n{humi_value}"  
            elif '/rain_value' in request:  
                response = f"HTTP/1.1 200 OK\nContent-Type:  
text/plain\n\n{rain_value}"  
            elif '/light_value' in request:  
                response = f"HTTP/1.1 200 OK\nContent-Type:  
text/plain\n\n{light_value}"  
            else:  
                html_webpage = webpage_outdoor(temp_value, humi_value,  
light_value, rain_value)  
                response = "HTTP/1.1 200 OK\nContent-Type:  
text/html\r\n\r\n" + html_webpage  
            [...]  
            led.off()  
        except OSError as e:  
            machine.reset()
```

Die Funktion `run_server` wurde leicht angepasst. In Abhängigkeit der Anfrage eines Clients sendet der Raspberry Pi Pico W verschiedene Antworten.

Falls ein Messwert angefragt wird, dann wird dieser als Text übermittelt. Wenn kein Messwert angefragt wird, dann wird die Webseite als Antwort gesendet. Als Zusatz blinkt auch hier bei jeder Anfrage eines Clients die eingebaute LED. Der Code zur Webseite ist auch nahezu identisch zur Webseite der anderen Station. Deshalb wird der HTML-Code nicht erneut aufgeführt und beschrieben.

Probleme und Verbesserungen

Ursprünglich hatte ich die Vision, dass nur eine Webseite existiert, die alle Daten der letzten 24 Stunden von allen Sensoren sammelt und in diverse Graphen darstellt. Hierbei kam der Raspberry Pi Pico W an seinen Grenzen und es traten einige Schwierigkeiten auf. Ein Problem war, dass die Queues nicht vollständig in den HTML-Code eingebunden werden konnten. Eine Queue war immer unvollständig. Deshalb fiel die Entscheidung, verschiedene Webseiten für beide Stationen zu erstellen. Hierbei hat auch nur die Webseite der Innenstation die Möglichkeit, die Daten in einem Graphen darzustellen, weil nur zwei Messwerte vorhanden sind. Die andere Station hat zu viele Daten für vier Diagramme.

Eine Lösung wäre, den Raspberry Pi Pico W der Innenstation mit einem Raspberry Pi 4 oder sogar 5 zu ersetzen. Dieser hätte genug Leistung, um alle Dateien in diversen Diagrammen darstellen zu können. Des Weiteren wird dann auch nur eine Webseite benötigt. Die Microcontroller hätten die Aufgabe, die Daten von den Sensoren zu lesen und über HTTP zur Verfügung zu stellen. Zudem könnte man auch mit weiteren Raspberry Pi Picos verschiedene Daten wie Temperatur oder Feuchtigkeit in weiteren Räumen messen. Man kann auch weitere Sensoren bspw. für den Luftdruck oder Luftqualität verwenden.

Zusätzlich hätte man auch die Möglichkeit dieses Projekt mit Home Automation zu verknüpfen. Da ich aber nicht viele Geräte bezüglich Home Automation besitze, konnte ich das Projekt nicht mit weiteren Geräten im Haushalt verbinden.

Grundlegend kann ich sagen, dass ich viel Spaß bei diesem Projekt hatte. Zukünftig könnte ich mir auch vorstellen, weitere Projekte mit Raspberry Pis oder andere Microcontroller zu beginnen.