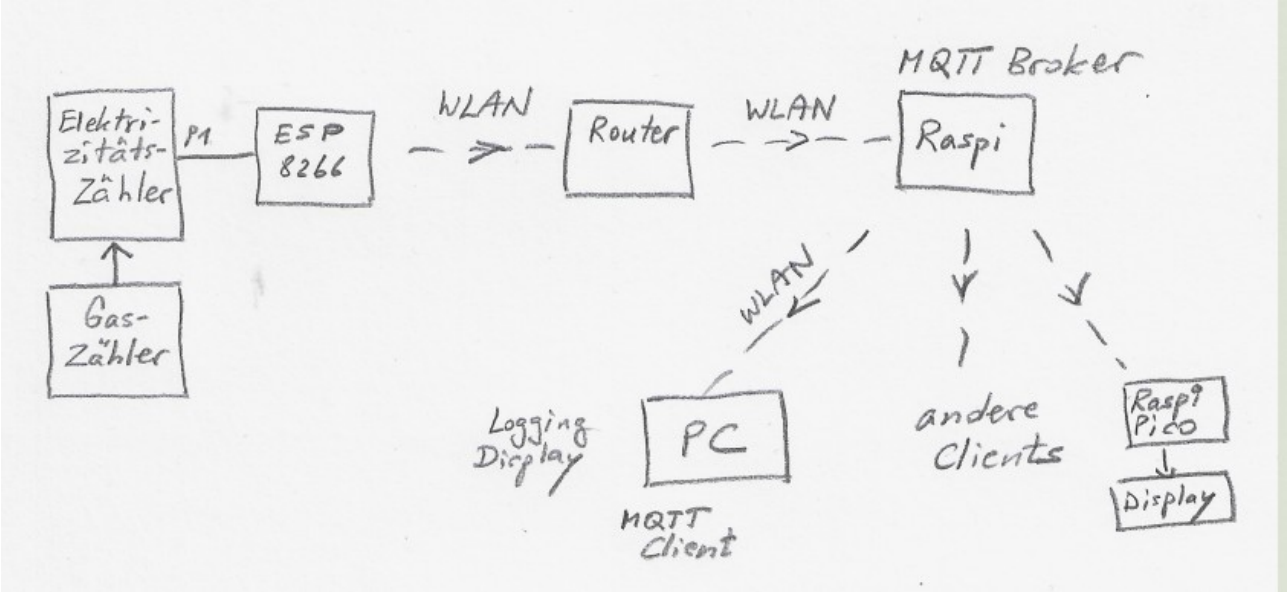
Energie sparen mit Smartyreader und MQTT

Von jean-claude.feltes@education.lu

Guy Weiler ist Lehrer im LTAM und ein Pionier der Erneuerbaren Energie. Durch ihn inspiriert habe ich begonnen, ein Energie-Monitoring-System aufzubauen, welches mir erlaubt, von irgendwo im Haus den aktuellen Energiverbrauch abzulesen. Das Ganze ohne Cloud-Anbindung, leider über WLAN, aber man könnte es auch über LAN machen.

Es war auch eine Gelegenheit, mich ein wenig in die Übertragung mit MQTT einzuarbeiten.

1. Übersicht



Mein “smarter” Elektrizitätszähler misst den Verbrauch der elektrischen Energie, die Daten des Gaszählers werden ebenfalls eingekoppelt. Prinzipiell ist es möglich, die Daten auf dem kleinen Display zu sehen, dazu muss ich aber in den Keller gehen und so lange Knöpchen drücken bis ich das gewünschte Menü gefunden hab.

All diese Daten stehen aber auch an der P1-Schnittstelle zur Verfügung:

https://www.weigu.lu/microcontroller/smartyReader_P1/index.html

Ein ESP8266 decodiert die Daten und sendet sie im MQTT-Protokoll über das WLAN-Netz zu einem Raspi, der die Rolle des MQTT-Brokers (MQTT Server) übernimmt.

Über WLAN können sich andere Geräte als MQTT Clients die gewünschten Daten abholen.

Momentan ist dies mein PC, später soll z.B. ein Raspi Pico mit einem Display die Daten in der Küche anzeigen, wo jeder sie bequem lesen kann.

2. MQTT

Die Daten werden mit Hilfe des MQTT-Protokolls übertragen. Dies ist ein einfach zu hanhabendes und übersichtliches Protokoll, welches Nachrichten überträgt, die immer aus einem “Topic” (Thema / Gerät / Zuordnung) und einer “Message” (Daten / Nachrichten) zum Thema bestehen:

https://www.weigu.lu/tutorials/sensors2bus/06_mqtt/index.html

Die Topics können hierarchisch organisiert sein z.B.:

```
smartyreader/volt_l1_V 229.00
smartyreader/volt_l2_V 229.00
smartyreader/volt_l3_V 229.00
```

Hier werden die Spannungen der 3 Phasen unter dem Topic “smartyreader” mit Unter-Topic “volt…” übertragen.

3. Auslesen und Übertragen der Zähler-Daten

Um auslesen zu können braucht man einen Code (die Daten sind verschlüsselt). Diesen habe ich über luxmetering.lu angefragt und ich konnte ihn dann ganz offiziell im Büro der Gemeinde-verwaltung abholen.

Weiter ist ein ESP8266 mit einer kleinen Zusatzbeschaltung (ein Widerstand und 2 Kondensatoren) erforderlich. Die Platine erhält man sehr günstig bei Guy Weiler, es würde aber auch eine Lochrasterplatine reichen.

Software für den ESP8266 gibt es hier:

https://github.com/weigu1/SmartyReader

Es müssen einige Parameter wie z.B. Netz-SSID, Passwort und IP-Adresse angepasst werden.

4. Der MQTT Broker

Auf dem Raspi muss die Broker-Software installiert werden. Ich benutze hier wie von Guy vorgeschlagen die freie Mosquitto-Software:

https://mosquitto.org/

Installation von der Kommandozeile:

```
sudo apt install mosquitto
```

Die Konfiguration erfolgt in der Datei `/etc/mosquitto/conf.d`

Dies kann man z.B. mit dem Texteditor nano erledigen:

```
cd /etc/mosquitto/conf.d
sudo nano mymosqui.conf
```

In diese (neu erzeugte) Datei werden die folgenden Zeilen eingefügt:

```
listener 1883
allow_anonymous true
```

Dies legt den Port 1883 fest (Standard), und man kann sich anonym einloggen.

Anschliessend startet man den Broker mit

```
sudo service mosquitto restart
```

Test:

```
sudo service mosquitto status
```

In der Meldung wird der Status “active running” angezeigt

```
● mosquitto.service - Mosquitto MQTT Broker
     Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2023-01-24 14:41:04 CET; 1h 34min ago
```

Wenn dies geklappt hat wird der Mosquitto server jedesmal nach dem Booten aktiviert.

5. MQTT clients: Kommandozeilentools

Von Mosquitto gibt es einen fertigen Client (Subscriber) , der so installiert wird:

```
sudo apt install mosquitto-clients
```

Um z.B. die aktuelle verbrauchte Gesamtleistung in Watt auszulesn, benutzt man

```
mosquitto_sub -t smartyreader/power_consumption_calc_from_energy_W
```

Das Topic “`smartyreader/power_consumption_calc_from_energy_W “`

ist hier relativ lang, so dass der Befehl ein wenig unübersichtlich aussieht.

Wenn alles funktioniert, werden die Leistungswerte fortlaufend ausgegeben:

```
pi@raspi3:~ $ mosquitto_sub -t smartyreader/power_consumption_calc_from_energy_W
3960.00
t3960.00
```

Nun könnte man diese Werte schon durch Anhängen von “> power.dat” in die Datei “power.dat” schreiben lassen und später schöne Grafiken davon erzeugen.

6. MQTT clients: Python Software

Am schönsten ist es natürlich, selbst was zu programmieren. Dann kann man alles anpassen so wie man will.

Eine einfache Python – Bibliothek für MQTT kann mit pip installiert werden:

pip3 install paho-mqtt (hier pip3 für Python3)

Programm zum Lesen aller gesendeten Daten

Im Internet findet man eine Menge Beispiele und Erklärungen, am besten sieht man sich aber wohl die Dokumentation von paho an:

https://pypi.org/project/paho-mqtt/

Hier die einfachste Möglichkeit:

```
from paho.mqtt import client as mqtt_client

broker = '192.168.179.26'        # Raspi as broker WLAN
port = 1883
topic_subscribe = "smartyreader/#"
client_id = "getsmarty_P1"
username = 'jcf'
password = 'public'

def on_connect(mqtt_client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT broker")
    else:
        print("Failed to connect with code ", rc)

    # Subscribing in on_connect() means that if we lose the connection and
    # reconnect then subscriptions will be renewed.
    client.subscribe(topic_subscribe)

def on_message(client, userdata, msg):
    topic = msg.topic
    message = msg.payload.decode()
    print(topic + "\t" + message)

client = mqtt_client.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect(broker, port)

client.loop_forever()
```

Zunächst werden die Callback-Funktionen on_connect und on_message definiert.

Die on_connect-Funktion meldet ob die Verbindung erfolgreich war oder nicht.
Hier wird auch die Subskription zum Topic gemacht. Das hat den Vorteil, dass diese im Falle eines reconnects erneuert wird.

Die on_message-Funktion printet das empfangene Topic und die zugehörige Nachricht.
Da topic_subscribe = "smartyreader/#" (# ist ein Multilevel – Wildcard) werden alle Nachrichten des Zählers ausgegeben.

Im Hauptprogramm passiert folgendes:

- Es wird ein Client instanziiert
- Die Callback-Funktionen werden dem Client mitgeteilt

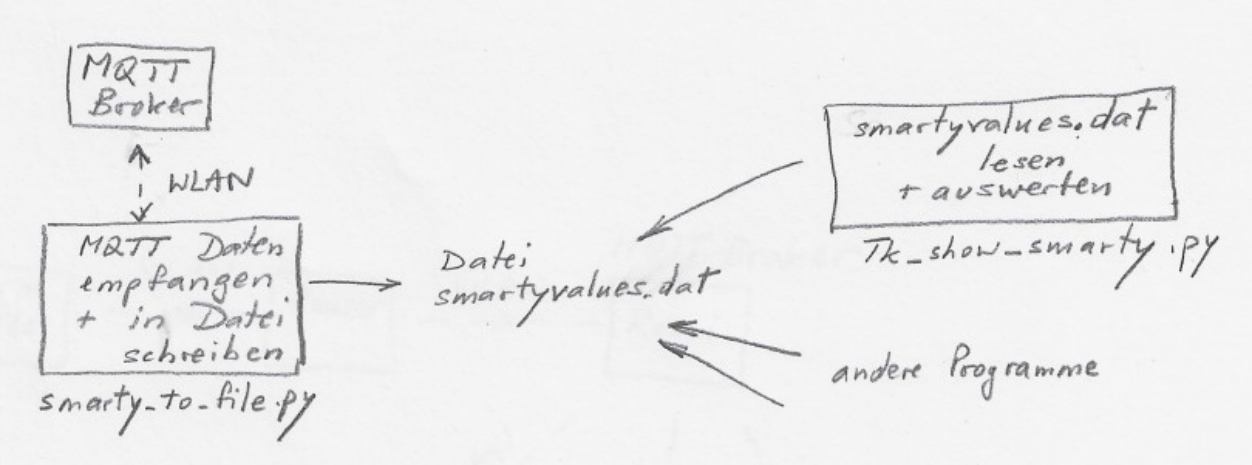
- Der Client verbindet sich mit dem Broker
- Das Ganze läuft in einer Endlosschleife

Wenn man dieses Grundgerüst hat, kann man es nach belieben ausbauen, z.B. die Daten filtern und nur die wichtigsten anzeigen, oder einge davon loggen usw.

Anzeige der Daten mit Tkinter

Hier gibt es eine Schwierigkeit: die MQTT-Schleife client.loop_forever ist blockierend, d.h. wir können nicht mit der Tkinter mischen.

Es gibt mehrere Möglichkeiten diese Schwierigkeiten zu überwinden, mir schien es am einfachsten die Sache zu modularisieren, so dass ein Script die Daten vom Broker liest und in eine Datei schreibt. In dieser Datei stehen dann die aktuellen Werte zur Verfügung, sie können von einem oder mehreren anderen Programmen ausgelesen und verwertet werden.



Das Programm **smarty_to_file.py** empfängt die Daten über MQTT und schreibt sie in eine Datei smartyvalues.dat

Hier braucht gegenüber dem vorherigen Programm nur die Funktion on_message geändert zu werden:

```
def on_message(client, userdata, msg):
    topic = msg.topic
    topic = topic.replace("smartyreader/", "")
    message = msg.payload.decode()

    print(topic + "\t" + message)
    write_all_to_file(topic, message)
```

Da die Topics alle mit “smartyreader/” beginnen, wird dieser irrelevante Teil gelöscht.

Die Funktion write_all_to_file() muss natürlich noch definiert werden:

```
file_is_open = False
filehandle = None
```

```
def write_all_to_file(topic, message):
    global file_is_open, filehandle

    if "ntp_datetime" in topic:     # first message
        print("Writing to ", filename)
        filehandle = open(filename, 'w')
        file_is_open = True


    if file_is_open:
        #f.write(time.now)
        #f.write('\t')
        filehandle.write(topic)
        filehandle.write('\t')
        filehandle.write(message)
        filehandle.write('\n')
        if "gas_consumption_calc_cumul_day" in topic:
            filehandle.close()
            file_is_open = False
```

Es gibt die Schwierigkeit, dass man nicht unbedingt weiss wann die Nachrichten beginnen und wann sie enden, es sind ja viele, und sie werden einzeln verschickt. Ich habe hier eine einfache Methode angewandt. Nach meiner Beobachtung enthielt die erste Nachricht immer die Zeitinformation, die letzte "gas_consumption_calc_cumul_day". Dies nutze ich hier aus um die Datei zu öffnen und zu schliessen.

Nachdem smarty_to_file.py läuft kann das Programm **tk_show_smarty.py** auf die Daten zugreifen und sie anzeigen.

Die Datei smartyvalues.dat braucht nur gelesen zu werden wenn sie sich geändert hat.

Um dies zu erkennen gibt es eine Funktion file_has_changed:

```
import os
file_time = os.stat(filename).st_mtime

def file_has_changed(filename):
    global file_time

    t = os.stat(filename).st_mtime
    if t != file_time:
        file_time = t
        return True
    else:
        return False
```

Es werden noch einige Hilfsfunktionen zum Verarbeiten definiert:

```
def get_values(filename):
    with open(filename, 'r') as f:
        s = f.read()

    return s


def extract (s, s_search):
    # Extract all lines containing s_search from s
    sr = ""
    lines = s.splitlines()
```

```
    for line in lines:
        if s_search in line:
            sr += line + "\n"
    return sr

def put_text(txtctrl, title, s, delete = True):
    # put title and textstring s into txtctrl
    if delete:
        txtctrl.delete(1.0, tk.END)
        txtctrl.insert(1.0, title + '\n')
    txtctrl.insert(tk.END, s)
```

Die Funktion watch_file erledigt den grössten Teil der Arbeit:

```
def watch_file():
    global i
    print(i)
    i += 1
    lbl.config(text = str(i))

    if file_has_changed(filename):
        i = 0
        print("File has changed")
        s = get_values(filename)

        if s:
            print(s)
            print()

            # all values:
            put_text(txt,"", s)
            '''
            txt.delete(1.0, tk.END)
            txt.insert(1.0, s)
            '''
            # time
            t = extract(s, "timestamp")
            tn = t.replace("timestamp", "")
            tnn = tn.split("T")
            print(tnn)
            ti = tnn[0]
            da = tnn[1]
            put_text(txttime, "Time:", ti + '\n')
            put_text(txttime, " ", da, delete = False)

            # currents:
            currs = extract(s, "curr")
            currs = currs.replace("curr_l", "L")
            put_text(txtcurr, "Currents:", currs)

            # voltages:
            volts = extract(s, "volt_l")
            volts = volts.replace("volt_l", "L")
            put_text(txtvolt, "Voltages:", volts)

            # power:
            p = extract(s, "act_pwr_imp_p_plus")
            pn = p.replace("act_pwr_imp_p_plus", "power_kW")
            pn = pn.replace("power_kW_l", "L")
            put_text(txtpowr, "Power:", pn)

            p = extract(s, "power_consumption_calc_W")
            pn = p.replace("power_consumption_calc_W", "power_total_W")
            put_text(txtpowr2, "Power:", pn)

            p = extract(s, "power_consumption_calc_m")
```

```
            pn = p.replace("power_consumption_calc_", "Ptot/")
            put_text(txtpowr2, "Power:", pn, delete = False)

    root.after(1000, watch_file)
```

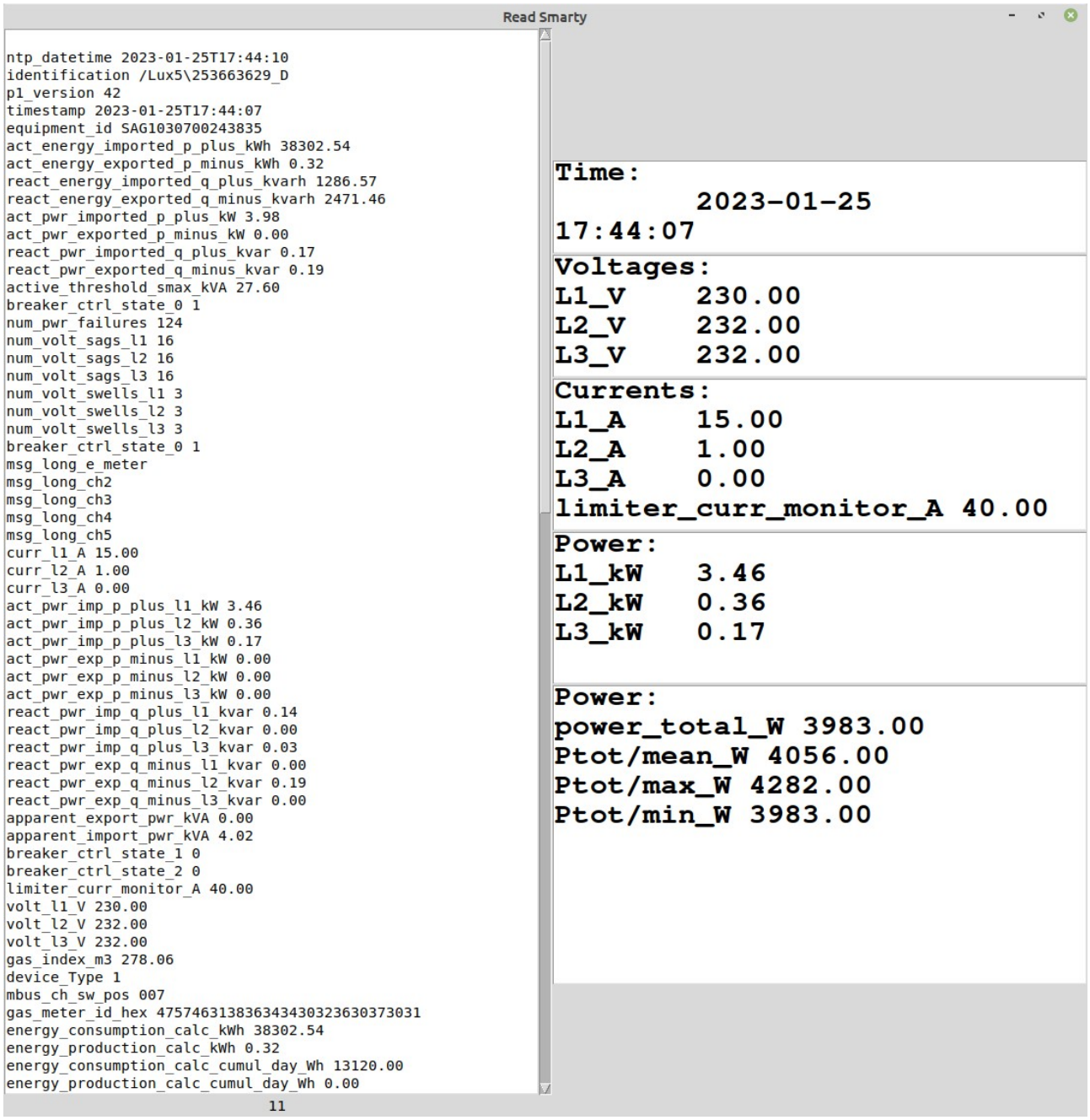
Es werden einige Umformungen gemacht, um die Darstellung übersichtlicher zu gestalten.

Mittels root_after() wird jede Sekunde einmal kontrolliert ob die Daten sich geändert haben.

Das Hauptprogramm definiert separate Textfelder um die Ergebnisse übersichtlich zu präsentieren

```
import time
import tkinter as tk
from tkinter.scrolledtext import ScrolledText
filename = "smartyvalues.dat"
i = 0
myfont = ("Courier", 20, "bold")

root = tk.Tk()
root.title("Read Smarty")
#root.protocol("WM_DELETE_WINDOW", fileexit)

# Left frame:
leftfr = tk.Frame(root)
txt =  ScrolledText(height=60,width=60, master=leftfr)
txt.pack()
lbl = tk.Label(leftfr, text = "0")
lbl.pack()
leftfr.pack(side = tk.LEFT)

# Right frame:
rightfr = tk.Frame(root)
rightfr.pack(side = tk.LEFT)

txttime = tk.Text(master = rightfr, height=3,width=30, font = myfont)
txttime.pack()

txtvolt = tk.Text(master = rightfr, height=4,width=30, font = myfont)
txtvolt.pack()

txtcurr = tk.Text(master = rightfr, height=5,width=30, font = myfont)
txtcurr.pack()

txtpowr = tk.Text(master = rightfr, height=5,width=30, font = myfont)
txtpowr.pack()

txtpowr2 = tk.Text(master = rightfr, height=10,width=30, font = myfont)
txtpowr2.pack()

watch_file()
root.mainloop()
```

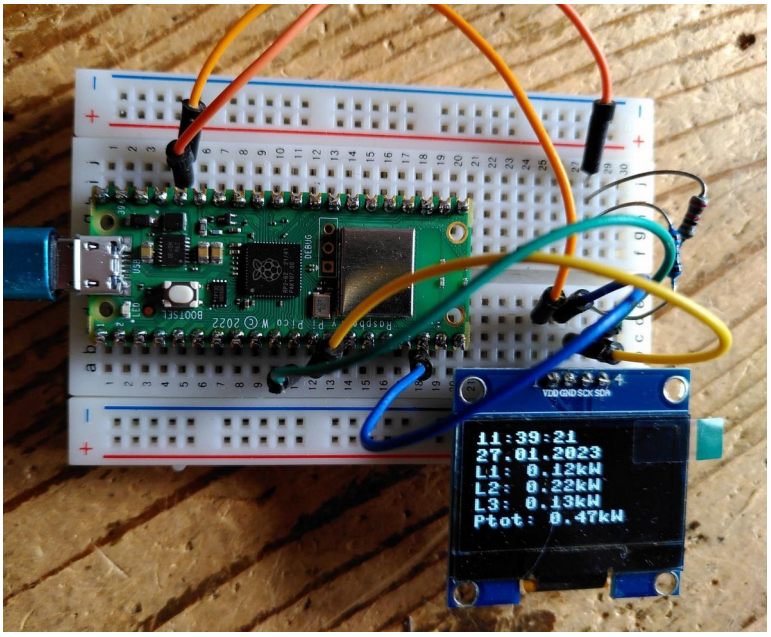
Das Ergebnis sieht dann so aus:



Auf der linken Seite werden alle Daten angezeigt, rechts nur die wichtigsten etwas grösser.

7. Ein Raspi Pico als Client

Mit diesen kann man irgendwo im Haus den Verbrauch kontrollieren.



Die Software ist in Micropython geschrieben.

In Thonny kann man unter “Tools – Manage packages” die MQTT-Bibliothek umqtt.simple installieren. Diese landet dann im Folder /lib.

Übrigens ist es eine gute Idee alle Bibliothen im /lib – Folder zu installieren. Dort werden sie problemlos gefunden.

Im Internet gibt es etliche Tutorials, z.B. hier:

https://www.tomshardware.com/how-to/send-and-receive-data-raspberry-pi-pico-w-mqtt

Die Vorgehensweise ist ähnlich wie bei den beschriebenen PC-Programmen, ausser dass zuerst noch die WLAN - Verbindung aufgebaut werden muss.

Diese hat mir übrigens einige Schwierigkeiten gemacht.
Unter Thonny war kein Problem, aber beim Starten im Standalone – Betrieb, nur mit einer 5V – Spannungsquelle verbunden, gab es immer wieder WLAN – Fehler und die Software blieb hängen.

Möglicherweise ist die Ursache ein Timing-Problem da einige Dinge bei der Kommunikation mit dem WLAN-Chip asynchron ablaufen.

Ich habe dieses Problem nun “quick and dirty” gelöst (na ja, so quick auch wieder nicht): im Falle eines Fehlers sowohl bei der WLAN- als auch bei der MQTT-Verbindung wird ein Reset ausgelöst. In der Praxis ging es dann so: WLAN ERROR – Reboot – WLAN OK.

Die Software kann hier gefunden werden:

https://github.com/jean-claudeF/smartyreader_pico