Automatisches Solar-Nachführungssystem

Abgabe 06.05.2025

Erstellt von: Fynn Bremer

Inhaltsverzeichnis

1. Projektbeschreibung

Das Projekt umfasst die Entwicklung eines automatisierten Nachführungssystems für ein Solarpanel, das sich optimal zur Sonne ausrichtet, um den Energieertrag zu maximieren. Die Ausrichtung erfolgt durch Auswertung von Lichtintensitäten mittels vier Lichtsensoren (LDRs), diese setzen dann die Schubmotoren in Kraft für die gewünschte Ausrichtung. Zusätzlich überwacht ein Windsensor die Windgeschwindigkeit, um bei Sturmgefahr das Panel automatisch in eine Schutzposition zu bringen.

Ein integriertes Dashboard zeigt in Echtzeit wichtige Umweltdaten (Temperatur, Außentemperatur, Luftfeuchtigkeit, Helligkeit, Windgeschwindigkeit) sowie aktuelle Leistungswerte (Spannung, Stromstärke, Leistung) an. Die Kommunikation erfolgt über das MQTT-Protokoll. Neben dem Automatikbetrieb ist eine manuelle Steuerung des Panels über MQTT-Befehle möglich. Das System verwendet u.a. folgende Sensoren und Module: DS18B20 (Temperatur), AHT10 (Luftfeuchtigkeit), INA226 (Spannung/Strommessung) und einen Reedkontakt für die Windmessung. Die Motorsteuerung erfolgt über BTS7960 Motortreiber.

2. Projektziel

Ziel des Projekts ist die Entwicklung eines intelligenten Systems, das ein Solarpanel automatisch zur Sonne ausrichtet, um die maximale Energieausbeute zu erzielen. Das System überwacht außerdem relevante Umweltdaten wie Temperatur, Luftfeuchtigkeit, Windgeschwindigkeit und Lichtintensität und stellt diese über ein MQTT-basiertes Dashboard in Echtzeit bereit.

Zusätzlich kann das Solarpanel manuell über MQTT-Kommandos gesteuert werden, um größtmögliche Flexibilität zu gewährleisten.

3. Funktionsumfang

Automatische Nachführung:

- Das Solarpanel wird auf Basis der Lichtstärke-Sensoren (LDRs) ausgerichtet, sodass es stets optimal zur Sonne zeigt.

Manuelle Steuerung:

- Manuelle Bewegung des Panels in X- und Y-Richtung über MQTT-Befehle möglich (z.B. für Wartung oder besondere Betriebsmodi).

Umweltdatenerfassung:

- Temperatur- und Luftfeuchtigkeitsmessung (DS18B20 und AHT10 Sensoren)
- Windgeschwindigkeitsmessung (Reed-Sensor)
- Lichtintensitätsmessung an vier Seiten (LDRs)
- Ermittlung von Spannung, Strom und Leistung (INA226-Modul)

Sturmschutz:

Überschreitet die Windgeschwindigkeit eine festgelegte Grenze, fährt das Panel automatisch in eine sichere Position und deaktiviert den Automatikbetrieb.

MQTT-Datenübertragung:

Alle erfassten Werte werden per MQTT an ein zentrales Dashboard gesendet. Steuerbefehle und Alarme werden ebenfalls über MQTT empfangen bzw. veröffentlicht.

Schwellenwertanpassung

Im Node-RED-Dashboard lassen sich verschiedene Schwellenwerte flexibel anpassen. So kann beispielsweise die maximal zulässige Windgeschwindigkeit festgelegt werden, bei deren Überschreitung das Solarmodul automatisch in eine Schutzstellung fährt.

Zudem lässt sich die Empfindlichkeit des Lichtsensors justieren, um bei gleicher Sonneneinstrahlung vergleichbare Messwerte zu erzielen – etwa zur Kompensation von Sensorabweichungen.

4. Technische Komponenten

Sensoren

- Microcontroller: ESP32-S3
- LDRs für Helligkeitserkennung
- DS18B20 Temperatursensor
- AHT10 Temperatur- und Feuchtigkeitssensor
- INA226 zur Strom- und Spannungsmessung
- Windsensor (Reedkontakt)

Motorsteuerung / Aktoren

- BTS7960 Motortreiber zur Ansteuerung der Bewegungsachsen (X und Y), Linear Schubstangen 300mm

Kommunikation

- WLAN-Verbindung zu einem lokalen Netzwerk
- MQTT-Protokoll für Messaging

Spannungsversorgung:

Die Sensorik, Aktuatoren und die Steuerungseinheit werden derzeit getrennt über eine 12V-Batterie mit Energie versorgt. Perspektivisch soll die Stromversorgung durch ein Netzteil erfolgen, sobald auf dem Gestell ein Solarpanel mit einer Leistung von etwa 600 W installiert ist. Der erzeugte Strom wird dann über einen Mikrowechselrichter direkt in das 230V-Netz eingespeist.

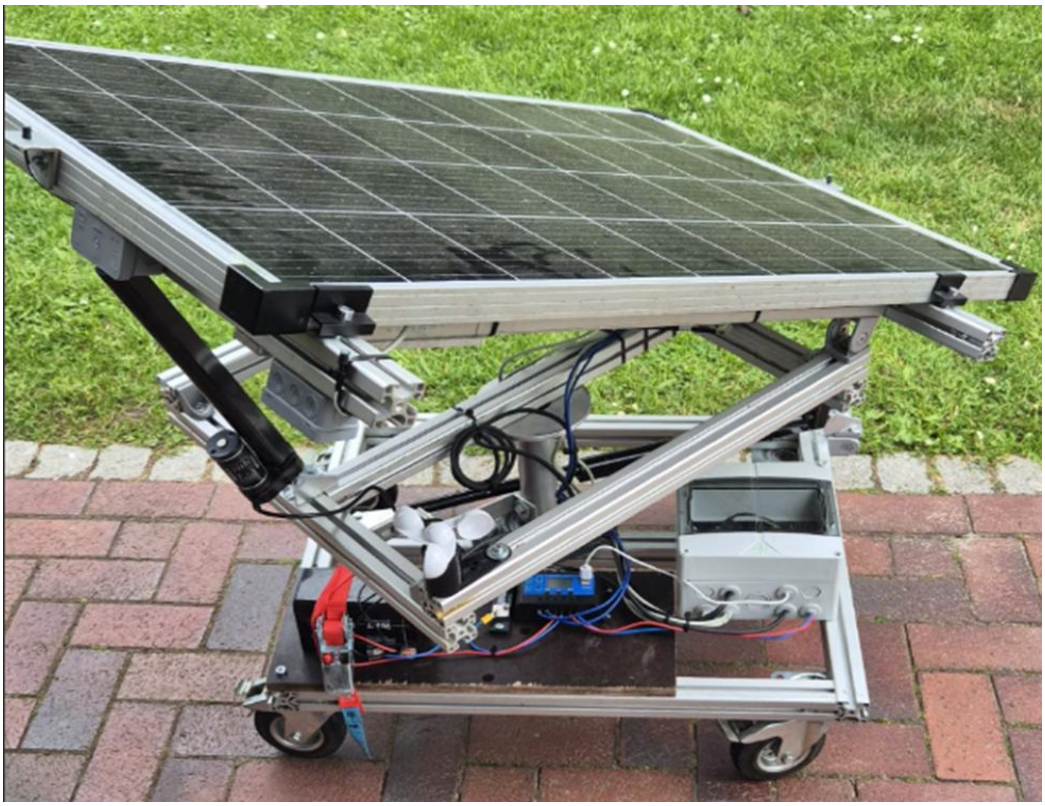
5. Ablauf und Steuerung

- Das System überwacht kontinuierlich die Lichtintensitäten der vier LDR-Sensoren.
- Es vergleicht die Werte und richtet das Panel durch gezielte Motorbewegungen automatisch zur besten Lichtquelle aus.
- Bei einer erkannten Sturmgefahr fährt das System das Panel ein und setzt eine Alarmmeldung ab. (für Testzwecke auf 15km/h gesetzt)
- Die Alarmmeldung muss erst quittiert, erst dann richtet sich das Panel wieder nach der Sonne aus.
- Eine manuelle Bedienung über MQTT ist jederzeit möglich und überschreibt temporär den Automatikmodus.

6. Aufbau und Verkabelung

Aufbau

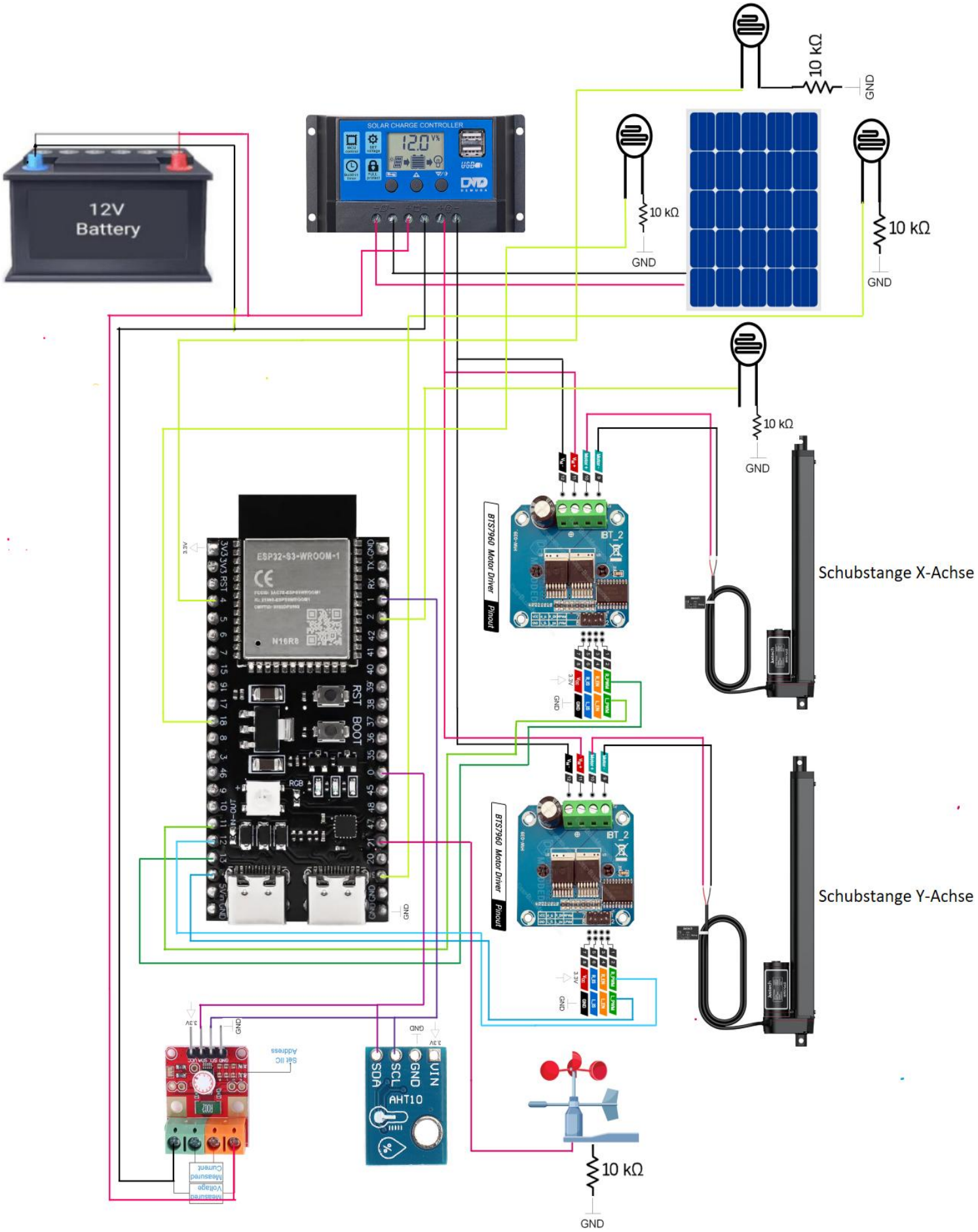
Das Gestell wurde aus ITEM-Aluprofilen gefertigt. Der große Vorteil dieses Materials liegt in seiner Witterungsbeständigkeit sowie der einfachen Verarbeitung und Montage. Für Testzwecke wurde das Gestell mobil auf Rollen ausgeführt, wohingegen es im regulären Einsatz fest im Boden verankert würde. Dabei würde auch die Technik bestmöglich vor äußeren Einflüssen geschützt werden.



Anschluss am Microcontroller

LDR Links	ADC Pin 18
LDR Rechts	ADC Pin 19
LDR Vorne	ADC Pin 2
LDR Hinten	ADC Pin 4
BTS7960 Motor X	PWM Pins 13 (RPWM) / 11 (LPWM)
BTS7960 Motor Y	PWM Pins 12 (RPWM) / 14 (LPWM)
DS18B20 (Temperatursenor Außen)	Pin 41 (OneWire)
AHT10 Sensor	I2C Pins SCL=1, SDA=0
INA226 Sensor	I2C Pins (gemeinsam mit AHT10)
Windsensor	Pin 21 (Interrupt Pin)

7. Schaltplan



8. Installations- und Bedienungsanleitung

1. Systemübersicht

Dieses System steuert die Ausrichtung eines Solarmoduls mithilfe von Lichtsensoren (LDR), Temperatur- und Windsensoren. Es verwendet MQTT zur Fernüberwachung und Steuerung. Bei zu hoher Windgeschwindigkeit fährt das Modul automatisch in eine Schutzposition.

2. Start und Verbindung

- Nach dem Einschalten verbindet sich das System mit dem fest eingestellten WLAN.
- Danach wird eine Verbindung zum MQTT-Broker 192.168.178.XX aufgebaut.
- Bei Problemen mit WLAN oder MQTT startet sich das Gerät neu.

3. MQTT-Kommandos / Manuelle Steuerung

Das System lässt sich auch manuell über MQTT-Befehle steuern. Diese Befehle können über einen MQTT-Client wie Node-RED, MQTT Explorer oder Home Assistant gesendet werden.

Verfügbare MQTT-Befehle:

Topic	Wert	Funktion
solar/befehl/bewege/x/links	–	Aktuator X fährt nach links (ausfahr)
solar/befehl/bewege/x/rechts	–	Aktuator X fährt nach rechts (einfahr)
solar/befehl/bewege/y/hoch	–	Aktuator Y fährt nach oben (ausfahr)
solar/befehl/bewege/y/runter	–	Aktuator Y fährt nach unten (einfahr)
solar/automatik	on / off	Automatikbetrieb ein- oder aus
solar/wind/zurücksetzen	-	Windalarm zurücksetzen
solar/wind/maximal	z. B. 12.0km/h	Erlaubte maximale Windgeschwindigkeit in km/h setzen
solar/ldr/links/Kalibrierung	z. B. 0.92	Korrekturfaktor für linken LDR setzen

4. Automatikbetrieb

- Ist solar/automatik aktiv und der Wind < Grenzwert, richtet sich das Modul automatisch nach dem hellsten Licht aus.
- Schwellenwert für Bewegung: Unterschied > 500 Einheiten zwischen zwei Sensoren.

5. Windalarm und Schutzfunktion

- Wenn die Windgeschwindigkeit > 15 km/h beträgt, wird Windalarm ausgelöst.
- Das Modul fährt automatisch in die Schutzstellung (beide Achsen einfahren).
- Automatikmodus wird deaktiviert und kann erst durch Senden von solar/wind/zurücksetzen reaktiviert werden.

6. Lichtsensor-Kalibrierung

- Falls der linke LDR-Sensor bei gleicher Helligkeit systematisch zu hohem oder zu niedrigem Wert liefert, kannst du dies per MQTT anpassen:
- Kalibrierung erfolgt über das Topic:
- solar/ldr/links/kalibrierung

Beispiel:

- Wenn der linke Sensor zu hohe Werte liefert, kannst du z. B. 0.90 senden.
- Das reduziert die Messwerte auf 90 % und bringt sie in Einklang mit dem rechten LDR.

Zulässiger Bereich:

- Werte zwischen 0.5 und 1.5 sind erlaubt.
- Die Korrektur wird sofort übernommen und wirkt sich auf die nächste Auswertung aus.

7. Wartungen & Störungen

Wartungen halbjährlich empfohlen

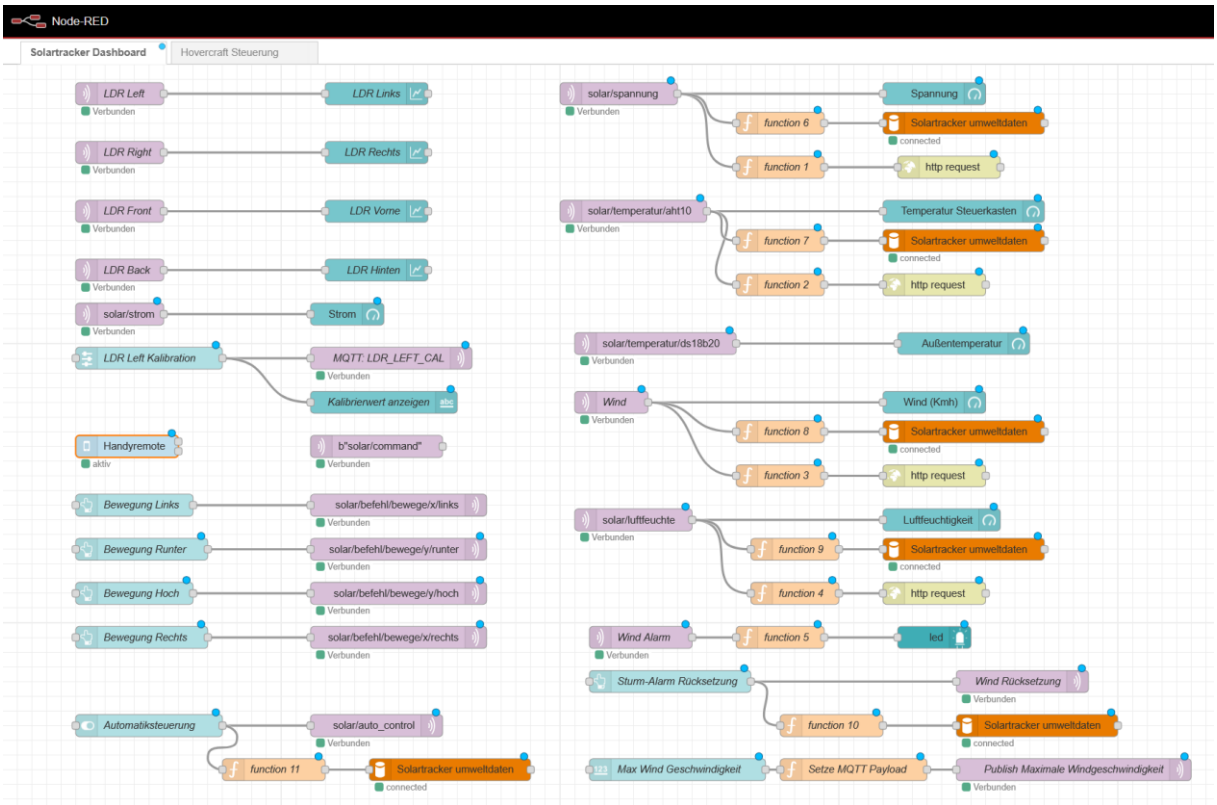
Wartungsbereich	**Maßnahme**
Lichtsensoren (LDRs)	Reinigen der Sensorabdeckungen mit trockenem Mikrofasertuch oder Druckluft.
Windsensor (Reedkontakt)	Auf festen Sitz prüfen, ggf. Kontaktreinigung mit Alkohol. Drehsensor prüfen.
Mechanik (Aktuatoren)	Sichtprüfung auf lose Kabel, Verschmutzung, Schmierung der Linearantriebe.
Kabelverbindungen	Sichtprüfung auf Korrosion, Wackelkontakte, Zugentlastung kontrollieren.
ESP32-Gehäuse	Staubfrei halten, Belüftung sicherstellen, ggf. Kondenswasser prüfen.

Störungshilfen

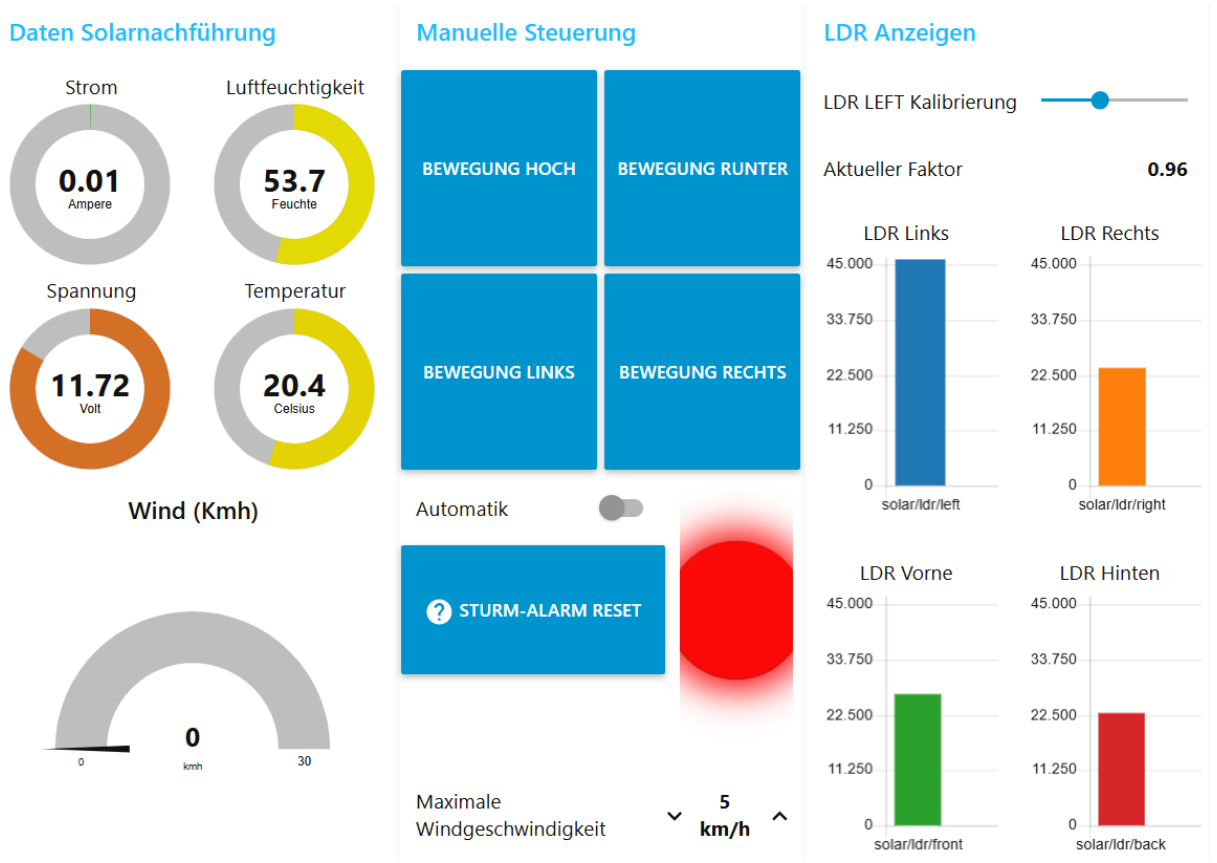
Problem	**Lösungsvorschlag**
Keine WLAN-Verbindung	Prüfen der SSID und Passwort
MQTT-Verbindung bricht ab	Serverstatus prüfen, Neustart System
Motoren reagieren nicht	Motorverkabelung und BTS7960 prüfen
Sensorwerte fehlen	Bus-Verkabelung und Sensoren prüfen
Ständige Windalarme	Sensor kalibrieren / reinigen

9. Node-RED Einbindung & Dashboard

Node-RED Einbindung



Node-RED Dashboard



10. Einbindung Thingspeak & HediSQL

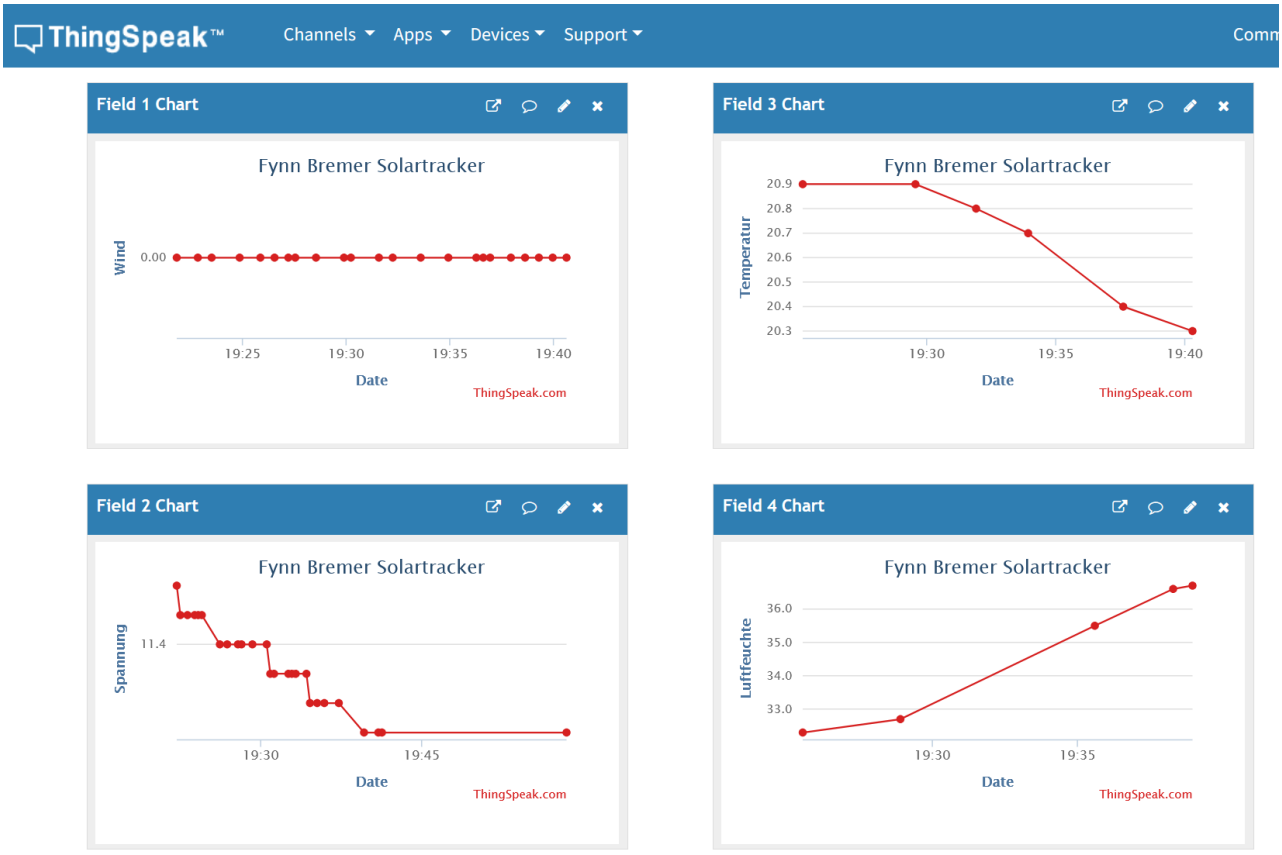
Fynn Bremer Solartracker

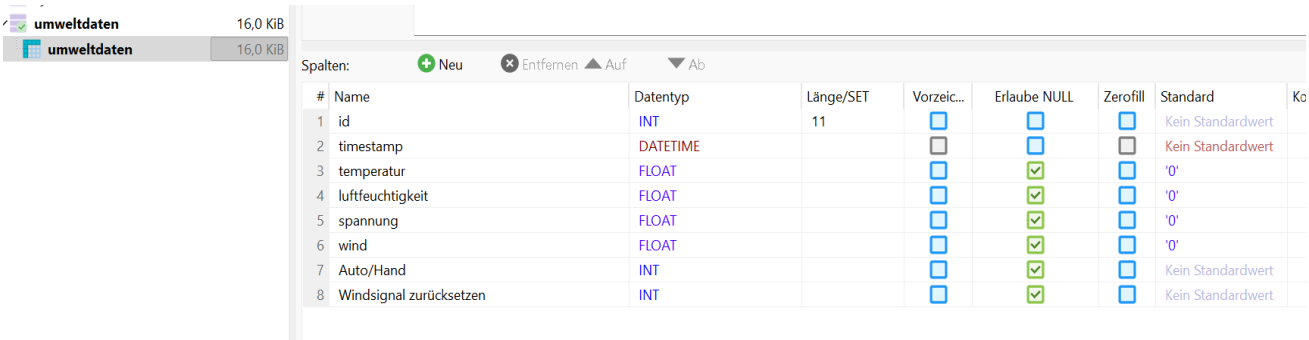
Channel ID: **2937118**
Author: mwa0000037422714
Access: Private

Aufnahme von Spannung, Temperatur, Luftfeuchte und Wind

Private View | Public View | Channel Settings | Sharing | API Keys | Data Import / Export

Add Visualizations | Add Widgets | Export recent data





#	Name	Datentyp	Länge/SET	Vorzeic...	Erlaube NULL	Zerofill	Standard	Ko
1	id	INT	11	☐	☐	☐	Kein Standardwert	
2	timestamp	DATETIME		☐	☐	☐	Kein Standardwert	
3	temperatur	FLOAT		☐	☑	☐	'0'	
4	luftfeuchtigkeit	FLOAT		☐	☑	☐	'0'	
5	spannung	FLOAT		☐	☑	☐	'0'	
6	wind	FLOAT		☐	☑	☐	'0'	
7	Auto/Hand	INT		☐	☑	☐	Kein Standardwert	
8	Windsignal zurücksetzen	INT		☐	☑	☐	Kein Standardwert	

11. Persönliches Fazit / Bilder

Die Entwicklung und Umsetzung dieses Projekts waren sowohl technisch herausfordernd als auch sehr spannend.

Es verbindet auf ideale Weise eine Mikrocontroller Steuerung, IoT-Kommunikation (MQTT) und Energieoptimierung, daher fand ich das Thema für mein Projekt sehr passend.

Besonders gut gelungen:

Flexibilität durch Automatik und manuelle Steuerung:

- Die Kombination ermöglicht ein Nutzererlebnis, wie man es von professionellen Systemen erwartet.

Integrierte Windüberwachung als Schutzmechanismus:

- Feature für die Langlebigkeit und Sicherheit des Systems.

Modernes Monitoring über MQTT-Dashboard:

- Nutzer können jederzeit aktuelle Umwelt- und Systemdaten einsehen und auf Veränderungen reagieren.

Kompakte Bauweise für begrenzte Flächen:

- Gerade für kleine Gärten oder Balkonanlagen bietet ein automatisch nachgeführtes Panel enorme Vorteile, da auf begrenztem Raum eine maximale Energieausbeute erzielt werden kann.

Herausforderungen:

Sensorabstimmung:

- Besonders die LDRs erforderten viel Arbeit bei der Kalibrierung, da sogar bei wenig Licht immer schnell der Maximalwert erreicht wurden ist. Erst durch das Einlöten eines richtigen Widerstandes sind die Werte bei voller Einstrahlung unter dem Maximalwert geblieben. Außerdem können Einflüsse wie Bewölkung oder Schmutz die Messergebnisse stark beeinflussen.

Feinabstimmung des Windsensors:

- Die Kalibrierung des Windsensors nahm auch einige Zeit in Anspruch, da ein richtiger Widerstand eingelötet werden musste, damit keine Extremen Schwankungen bei den Werten Zustandekommen. Außerdem hat der Windmesser pro Umdrehung drei Reedkontakt Signale ausgegeben, was immer für zu hohe Werte gesorgt hat, da im Programmcode mit einer Umdrehung gerechnet wurden ist.

Kommunikationsprotokolle und Timing-Probleme:

- Die gleichzeitige Integration von OneWire, I2C und WLAN auf einem Mikrocontroller brachte anfangs Timing-Schwierigkeiten mit sich, die gelöst werden mussten.

Was ich daraus gelernt habe:

Systemdenken:

- Projekte dieser Komplexität verlangen, dass man Hardware, Software und Umgebung als ein zusammenhängendes System betrachtet.

Fokus auf Robustheit:

- Funktion allein genügt nicht – insbesondere bei Outdoor-Anwendungen ist Zuverlässigkeit entscheidend.

Synergien von IoT und Energiegewinnung:

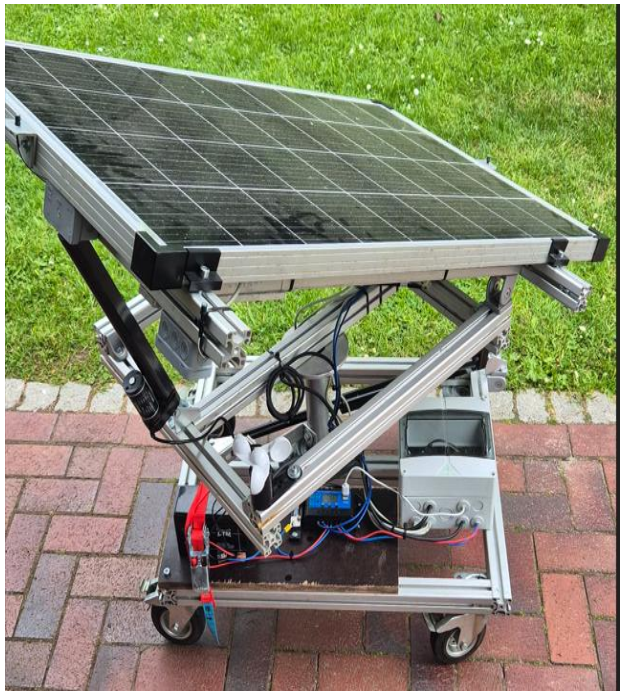
- Die intelligente Steuerung erneuerbarer Energiequellen wird in Zukunft eine immer zentralere Rolle spielen.

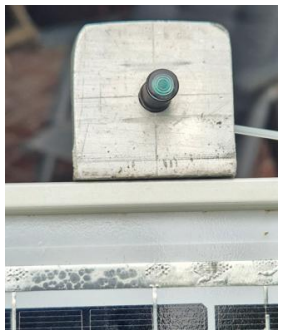
Marktpotenzial:

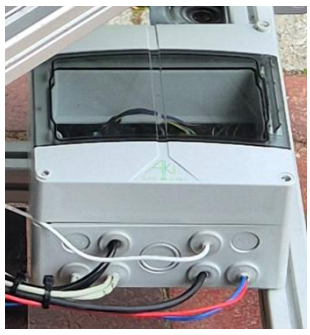
Automatisch nachgeführte, kompakte Solarsysteme könnten zukünftig eine echte Marktlücke schließen – speziell Balkonanlagen.

Wenn diese Systeme in Zukunft noch günstiger produziert werden können, könnten sie eine breite Anwendung im privaten Bereich finden und den Eigenverbrauch von Solarstrom deutlich steigern.

Bilder/ verwendete Materialen:







12. Vollständiger Quellcode / GitHub Link

https://github.com/Fynn0705/Solarnachf-hrungssystem-.git

```
#Automatisches Nachführungssystem für ein Solarpanel, das durch Lichtstärken Sensoren die
#Optimale Ausrichtung des Solarpanels findet
#Damit den Maximalen Energie Ertrag aus dem Solarpanel erzielt.
#Solarpanel lässt sich auch manuell verfahren.
#Bei einer zu hohen Windgeschwindigkeit fährt das Solarpanel in Schutzstellung.
#Temperatur und Helligkeit sowie Strom, Spannung und Leistung werden in einem Dashboard
#dargestellt.
#Ersteller Fynn Bremer
#letztes Update 04.05.2025

import machine
from machine import Pin, ADC, I2C
from time import sleep, ticks_ms, ticks_diff
import network
import onewire, ds18x20
from umqtt.simple import MQTTClient
from ina226 import INA226

# WLAN-Konfiguration
WLAN_NAME = "FRITZ!Box 6590 Cable Bremer"       # SSID des WLANs
WLAN_PASSWORT = "Maunzi2216"                    # Passwort für das WLAN

# MQTT-Konfiguration
MQTT_SERVER = "192.168.178.24"                  # Adresse des MQTT-Brokers
MQTT_CLIENT_ID = "solar_tracker"                # Eindeutige Gerätekennung

# MQTT-Themen (Topics)
MQTT_TOPIC_BEFEHL = b"solar/befehl"             # Steuerbefehle
MQTT_TOPIC_TEMP_DS18B20 = b"solar/temperatur/ds18b20" #Temperatur Außen
MQTT_TOPIC_TEMP_AHT10   = b"solar/temperatur/aht10" # Temperatur im Schaltkasten
MQTT_TOPIC_LUFTFEUCHTE = b"solar/luftfeuchte"   # Luftfeuchte vom AHT10
MQTT_TOPIC_LEISTUNG = b"solar/leistung"         # Elektrische Leistung
MQTT_TOPIC_SPANNUNG = b"solar/spannung"         # Spannung (INA226)
MQTT_TOPIC_STROM = b"solar/strom"               # Strom (INA226)
MQTT_TOPIC_WIND = b"solar/wind"                 # Windgeschwindigkeit
MQTT_TOPIC_LDR_LINKS = b"solar/ldr/links"       # Lichtsensor links
MQTT_TOPIC_LDR_RECHTS = b"solar/ldr/rechts"     # Lichtsensor rechts
MQTT_TOPIC_LDR_VORNE = b"solar/ldr/vorne"       # Lichtsensor vorne
MQTT_TOPIC_LDR_HINTEN = b"solar/ldr/hinten"     # Lichtsensor hinten
MQTT_TOPIC_AUTOMATIK = b"solar/automatik"       # Automatikbetrieb ein/aus
MQTT_TOPIC_WIND_ALARM = b"solar/wind/alarm"     # Windalarmstatus
MQTT_TOPIC_WIND_RESET = b"solar/wind/zuruecksetzen" # Rücksetzen des Windalarms
MQTT_TOPIC_WIND_MAX = b"solar/wind/maximal"     # Neue Grenzwerte setzen
MQTT_TOPIC_LDR_LINKS_KAL = b"solar/ldr/links/kalibrierung" # LDR-Korrekturfaktor

# Wind- und Kalibriergrenzwerte
maximale_windgeschwindigkeit = 10.0             # obere Grenze für Automatikbetrieb
WIND_ALARM_GRENZE = 15.0                        # Windgeschwindigkeit für Alarm
```

```
LDR_LINKS_KALIBRIERUNG = 0.95                # Korrekturwert für linken LDR


# Zeitverfolgung für Sensorintervall
letzte_sensorzeit = ticks_ms()

# Initialisierung des DS18B20 Temperatursensors
ds_pin = Pin(41)
ds_sensor = ds18x20.DS18X20(onewire.OneWire(ds_pin))
roms = ds_sensor.scan()

# LDR-Sensoren an ADC-Pins
LDR_LINKS = ADC(2)
LDR_RECHTS = ADC(4)
LDR_VORNE = ADC(18)
LDR_HINTEN = ADC(19)

# Steuerung für Aktuatoren (BTS7960 H-Brücke)
RPWM_X = Pin(14, Pin.OUT)
LPWM_X = Pin(12, Pin.OUT)
RPWM_Y = Pin(13, Pin.OUT)
LPWM_Y = Pin(11, Pin.OUT)

# Windsensor – Impulse zählen
WIND_PIN = Pin(21, Pin.IN, Pin.PULL_UP)
wind_impulse = 0
letzte_windzeit = 0

# IRQ-Handler für Windimpulse
def wind_callback(pin):
    global wind_impulse, letzte_windzeit
    jetzt = ticks_ms()
    if ticks_diff(jetzt, letzte_windzeit) > 50:  # Entprellung
        wind_impulse += 1
        letzte_windzeit = jetzt

WIND_PIN.irq(trigger=Pin.IRQ_FALLING, handler=wind_callback)

# Berechnung der Windgeschwindigkeit aus Impulsen
def berechne_windgeschwindigkeit(impulse, intervall_s):
    rpm = impulse * (30 / intervall_s)
    kmh = rpm * 0.1
    return kmh

# I2C-Bus & Sensoren initialisieren
i2c = machine.I2C(scl=Pin(1), sda=Pin(0), freq=100000)
AHT10_ADRESSE = 0x38
i2c.writeto(AHT10_ADRESSE, bytearray([0xE1, 0x08, 0x00]))  # Initialisieren
sleep(0.05)
    ina226 = INA226(i2c, 0x40)  # Strom/Spannungssensor INA226

    # WLAN-Verbindung aufbauen
    wlan = network.WLAN(network.STA_IF)
```

```
104.    wlan.active(True)
105.    if not wlan.isconnected():
106.        print(f"Verbinde mit WLAN {WLAN_NAME}...")
107.        wlan.connect(WLAN_NAME, WLAN_PASSWORT)
108.        timeout = 10
109.        while not wlan.isconnected() and timeout > 0:
110.            sleep(1)
111.            timeout -= 1
112.            print(".")
113.        if wlan.isconnected():
114.            print("WLAN verbunden, IP-Adresse:", wlan.ifconfig()[0])
115.        else:
116.            print("WLAN fehlgeschlagen!")
117.            machine.reset()
118.
119.    # MQTT-Verbindung und Abonnements
120.    def mqtt_verbinden():
121.        global client
122.        client = MQTTClient(MQTT_CLIENT_ID, MQTT_SERVER)
123.        client.set_callback(mqtt_callback)
124.        client.connect()
125.        client.subscribe(MQTT_TOPIC_BEFEHL + b"/#")
126.        client.subscribe(MQTT_TOPIC_AUTOMATIK)
127.        client.subscribe(MQTT_TOPIC_WIND_RESET)
128.        client.subscribe(MQTT_TOPIC_WIND_MAX)
129.        client.subscribe(MQTT_TOPIC_LDR_LINKS_KAL)
130.        print(" MQTT verbunden.")
131.
132.    automatik_aktiv = True
133.    wind_aktiv = False
134.    windgeschwindigkeit = 0.0
135.
136.    # Steuerung der Motoren über Richtungsparameter
137.    def bewege_aktor(r_pwm, l_pwm, richtung, dauer=0.5):
138.        r_pwm.value(0)
139.        l_pwm.value(0)
140.        sleep(0.1)
141.        if richtung == "ausfahren":
142.            r_pwm.value(1)
143.            l_pwm.value(0)
144.        elif richtung == "einfahren":
145.            r_pwm.value(0)
146.            l_pwm.value(1)
147.        sleep(dauer)
148.        r_pwm.value(0)
149.        l_pwm.value(0)
150.
151.    # MQTT-Ereignisbehandlung
152.    def mqtt_callback(topic, msg):
153.        global automatik_aktiv, wind_aktiv, maximale_windgeschwindigkeit
154.        print(f" Nachricht: {topic.decode()} - {msg.decode()}")
155.
156.        if topic == MQTT_TOPIC_AUTOMATIK:
```

```
            automatik_aktiv = msg.decode().lower() == "on"
            print("Automatik:", "EIN" if automatik_aktiv else "AUS")

        if topic == b"solar/befehl/bewege/x/links":
            bewege_aktor(RPWM_X, LPWM_X, "ausfahren", 10)
        elif topic == b"solar/befehl/bewege/x/rechts":
            bewege_aktor(RPWM_X, LPWM_X, "einfahren", 10)
        elif topic == b"solar/befehl/bewege/y/hoch":
            bewege_aktor(RPWM_Y, LPWM_Y, "ausfahren", 10)
        elif topic == b"solar/befehl/bewege/y/runter":
            bewege_aktor(RPWM_Y, LPWM_Y, "einfahren", 10)

        if topic == MQTT_TOPIC_WIND_RESET and wind_aktiv:
            print("Wind-Reset empfangen - Automatik reaktiviert.")
            automatik_aktiv = True
            wind_aktiv = False
            client.publish(MQTT_TOPIC_WIND_ALARM, b"OK")

        if topic == MQTT_TOPIC_WIND_MAX:
            try:
                neue_grenze = float(msg.decode())
                maximale_windgeschwindigkeit = neue_grenze
                print(f"Neue max. Windgeschwindigkeit: {maximale_windgeschwindigkeit} km/h")
            except ValueError:
                print("Ungültiger Wert für Windgrenze:", msg)

        if topic == MQTT_TOPIC_LDR_LINKS_KAL:
            try:
                neuer_faktor = float(msg.decode())
                if 0.5 <= neuer_faktor <= 1.5:
                    global LDR_LINKS_KALIBRIERUNG
                    LDR_LINKS_KALIBRIERUNG = neuer_faktor
                    print(f" Kalibrierung LDR links: {LDR_LINKS_KALIBRIERUNG:.2f}")
                else:
                    print(" Kalibrierungswert außerhalb von 0.5–1.5!")
            except ValueError:
                print(" Ungültiger Kalibrierungswert:", msg)

    mqtt_verbinden()

    # Temperatur und Luftfeuchte vom AHT10 auslesen
    def lese_aht10():
        try:
            i2c.writeto(AHT10_ADRESSE, b'\xAC\x33\x00')
            sleep(0.1)
            daten = i2c.readfrom(AHT10_ADRESSE, 6)

            if daten[0] & 0x08 != 0x08:
                print("Sensor nicht bereit")
                return

            roh_feuchte = ((daten[1] << 16) | (daten[2] << 8) | daten[3]) >> 4
            roh_temp = ((daten[3] & 0x0F) << 16) | (daten[4] << 8) | daten[5]
```

```
210.
211.         feuchte = round((roh_feuchte / 1048576) * 100, 1)
212.         temperatur = round((roh_temp / 1048576) * 200 - 50, 1)
213.
214.         print(" Temperatur:", temperatur, "°C  Feuchte:", feuchte, "%")
215.         client.publish(MQTT_TOPIC_TEMP_AHT10, str(temperatur))
216.         client.publish(MQTT_TOPIC_LUFTFEUCHTE, str(feuchte))
217.       except Exception as e:
218.         print("Fehler AHT10:", e)
219.
220.   # Hauptschleife: Sensoren lesen, MQTT verarbeiten, Motoren steuern
221.   while True:
222.     if not wlan.isconnected():
223.       print(" WLAN getrennt – Neustart...")
224.       sleep(5)
225.       machine.reset()
226.
227.     try:
228.       client.check_msg()  # Neue MQTT-Nachrichten verarbeiten
229.     except Exception as e:
230.       print(" MQTT-Fehler:", e)
231.       sleep(5)
232.       machine.reset()
233.
234.     jetzt = ticks_ms()
235.
236.     if ticks_diff(jetzt, letzte_sensorzeit) >= 20_000:
237.       # Temperatur DS18B20
238.       for rom in roms:
239.         temperatur = ds_sensor.read_temp(rom)
240.         print(f" DS18B20: {temperatur:.2f} °C")
241.         client.publish(MQTT_TOPIC_TEMP_DS18B20, f"{temperatur:.2f}")
242.
243.       # INA226 auslesen
244.       try:
245.         spannung = ina226.bus_voltage
246.         shunt_spannung = ina226.shunt_voltage
247.         shunt_widerstand = 0.002
248.         strom = shunt_spannung / shunt_widerstand
249.         leistung = ina226.power
250.         print(f" Spannung: {spannung:.2f} V, Strom: {strom:.2f} A, Leistung: {leistung:.2f} W")
251.         client.publish(MQTT_TOPIC_SPANNUNG, f"{spannung:.2f}")
252.         client.publish(MQTT_TOPIC_STROM, f"{strom:.2f}")
253.         client.publish(MQTT_TOPIC_LEISTUNG, f"{leistung:.2f}")
254.       except Exception as e:
255.         print("Fehler INA226:", e)
256.
257.       # Windgeschwindigkeit berechnen
258.       windgeschwindigkeit = berechne_windgeschwindigkeit(wind_impulse, 10)
259.       print(f" Windgeschwindigkeit: {windgeschwindigkeit:.2f} km/h")
260.       client.publish(MQTT_TOPIC_WIND, f"{windgeschwindigkeit:.2f}")
261.
262.       # Windalarm auslösen
```

```
            if windgeschwindigkeit > WIND_ALARM_GRENZE:
                print("STURMALARM! Wind zu stark:", windgeschwindigkeit)
                client.publish(MQTT_TOPIC_WIND_ALARM, f"ALARM: {windgeschwindigkeit:.2f}
    km/h")
                automatik_aktiv = False
                wind_aktiv = True
                bewege_aktor(RPWM_X, LPWM_X, "einfahren", 25)
                bewege_aktor(RPWM_Y, LPWM_Y, "einfahren", 25)

            wind_impulse = 0
            lese_aht10()
            letzte_sensorzeit = jetzt
            client.publish(MQTT_TOPIC_WIND_ALARM, b"ALARM" if wind_aktiv else b"OK")

        # Lichtsensoren auslesen und senden
        l_links = LDR_LINKS.read_u16() * LDR_LINKS_KALIBRIERUNG
        l_rechts = LDR_RECHTS.read_u16()
        l_vorne = LDR_VORNE.read_u16()
        l_hinten = LDR_HINTEN.read_u16()

        client.publish(MQTT_TOPIC_LDR_LINKS, str(l_links))
        client.publish(MQTT_TOPIC_LDR_RECHTS, str(l_rechts))
        client.publish(MQTT_TOPIC_LDR_VORNE, str(l_vorne))
        client.publish(MQTT_TOPIC_LDR_HINTEN, str(l_hinten))

        # Automatische Nachführung aktivieren, falls erlaubt
        if automatik_aktiv and windgeschwindigkeit < maximale_windgeschwindigkeit:
            if l_links > l_rechts + 500:
                bewege_aktor(RPWM_X, LPWM_X, "ausfahren", 2)
            elif l_rechts > l_links + 500:
                bewege_aktor(RPWM_X, LPWM_X, "einfahren", 2)

            if l_vorne > l_hinten + 500:
                bewege_aktor(RPWM_Y, LPWM_Y, "ausfahren", 2)
            elif l_hinten > l_vorne + 500:
                bewege_aktor(RPWM_Y, LPWM_Y, "einfahren", 2)

        sleep(0.2)
```