Dokumentation

IoT 2 (MECH-B-3-IOT-IOT2-ILV)

Abschlussprojekt

Bachelorstudiengang - Mechatronics, Design & Innovation

3. Semester

Lector: Dr. rer. pol. Julian Huber

Group: BA-MECH-22-IOT

Authors: Stefan Posch, Sandro Streicher, Elias Zischg

19. Februar 2024

Inhaltsverzeichnis

1 Einleitung

Das Projekt zielt darauf ab, eine Tageslichtsteuerung und Konstantlichtregelung zu implementieren. Dabei soll ein Miniaturmodell zur Veranschaulichung gebaut werden. Außerdem sollen die Parametrierung sowie einige Eingaben über den MQTT-Broker definiert werden können.
Die Tageslichtsteuerung orientiert sich dabei stark an der DIN-Norm. Die Konstantlichtregelung weist aufgrund der höheren Komplexität einige Abweichungen dazu auf - die Implementierung dient nur zur Veranschaulichung für die LV.
Folgender Use-Case wird angestrebt. Die befindlichen Aktionen im Code werden durch den System Controller ausgeführt:

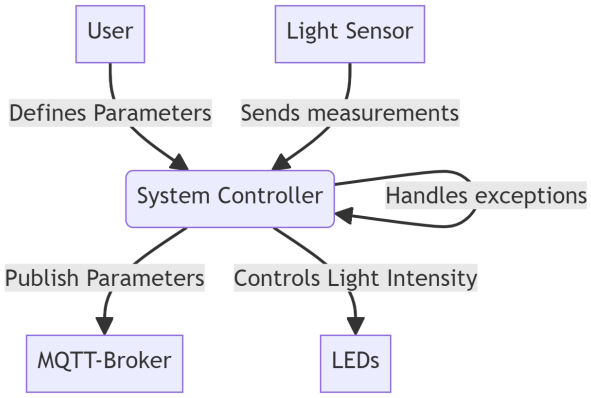

Abbildung 1.1: Use-Case des Projekts

Hier ist anzumerken, dass die Parameterdefinition über den MQTT-Broker auf Circuitpython aufgrund fehlender Bibliotheken nicht möglich ist. Größtenteils wurden die Parameter lokal im Code festgelegt - die erforderlichen Topics wurden dennoch angelegt. Leider konnten wir nicht überprüfen ob sich bei Änderung auf dem MQTT-Server die lokalen Variablen ändern. Weitere Details sind im Code beschrieben.

2 Kennlinie ADC(lx)

Um den Phototransistor TEMT6000 als Luxmeter zu verwenden, ist es notwendig, zunächst eine Kennlinie zu erstellen. Diese Kennlinie wurde unter Verwendung eines MX-ELEKTRONIK Mini-Luxmeters erstellt. Dabei wurde der TEMT6000 zusammen mit dem kalibrierten Luxmeter an einem identischen Messpunkt platziert, wobei beide Geräte die gleiche Ausrichtung hatten (Normalvektor 90° zum Boden). Anschließend wurden mehrere Messpunkte bei verschiedenen Lichtverhältnissen aufgezeichnet.
Folgenden Messpunkte wurden erfasst:

Lux/lx	**ADC**
0.44	524.64
39.4	2197.44
175	6345.48
270	9169.52
320	10449.8
463	14886.7
650	19695.54
720	21934.0
1600	45232

Tabelle 2.1: Messwerte der Kennlinie

Mithilfe eines Python-Skriptes wird ein sogenanntes curve fitting durchgeführt. Dieses ist notwendig, damit die Kennlinie als Funktion dargestellt werden kann. In diesem konkreten Fall wird ein linearer Fit mit 5%-iger Fehlertoleranz angestrebt:

Listing 1: Fitting a linear function to data

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
from numpy import loadtxt
import math as math



y = np.array([0.44, 39.4, 175, 270, 320, 463, 650, 720, 1600])
x = np.array([524.64, 2197.44, 6345.48, 9169.52,
              10449.8, 14886.7, 19695.54, 21934.0, 45232])
yer = np.array([0.044, 3.94, 17.5, 27.0, 32.0, 46.3, 65.0, 72.0, 160])
xer = np.array([52.464, 219.744, 634.548, 916.952,
                1044.98, 1488.67, 1969.554, 2193.40, 4523.2])



def lin (x,a,b):
    return a*x+b



plt.errorbar(x,y,yer,xer,fmt=".",elinewidth=0.5,capsize=2)
initvals= [1,1]
bestval,covar=curve_fit(lin,xdata=x,ydata=y,p0=initvals,sigma=yer+xer
,absolute_sigma=True)
xfit=np.linspace(0,46000,1000)
yfit=lin(xfit,*bestval)


plt.plot(xfit,yfit,label=str(bestval[0])+'x'+ str(bestval[1]))
plt.xlabel('ADC')
plt.ylabel('lux')


sigma = np.sqrt(np.diag(covar))


plt.legend(['linearer_Fit_mit_y=ax+b'])
plt.text(100, 1500, 'a=0.032;b=-17.166')
plt.savefig('lux_ADC_plot')
plt.show()


print(bestval)
print (sigma)
```

Folgende Abbildung 2.1 zeigt den curve fit. In orange ist die gefittete Funktion zu sehen. In blau sind die Messpunkte mit einer Fehlertoleranz von 5% zu sehen. Auf der y-Achse is die Beleuchtungsstärke abgebildet und auf der x-Achse der eingelesene ADC-Wert:

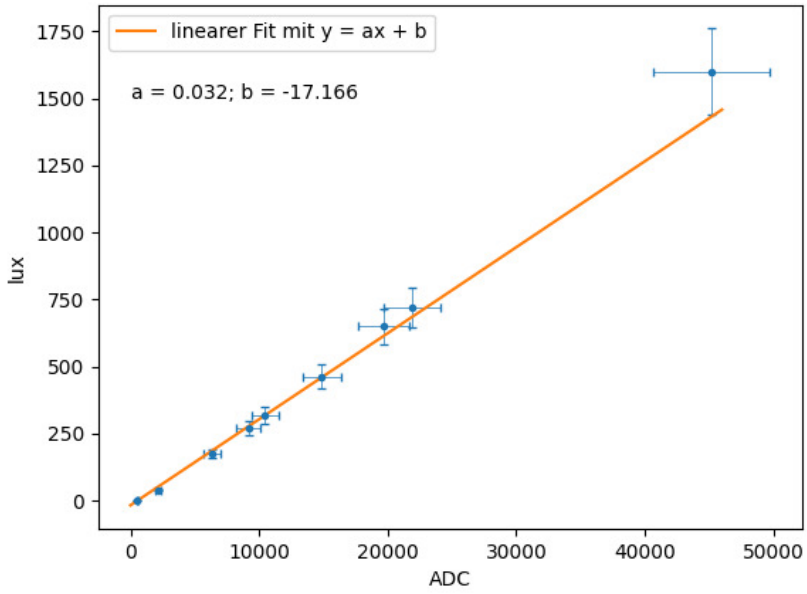

Abbildung 2.1: Curvefit: $Lux = 0,032 \cdot ADC - 17,166$

Im linken Plot der Abbildung 2.2 ist der Fotostrom als Funktion der Beleuchtungsstärke abgebildet. Hierbei ist ein linerare Zusammenhang zu erkennen. Im rechten Plot ist die relative Strahlungsempfindlichkeit als Funktion der Winkelverschiebung aufgetragen.

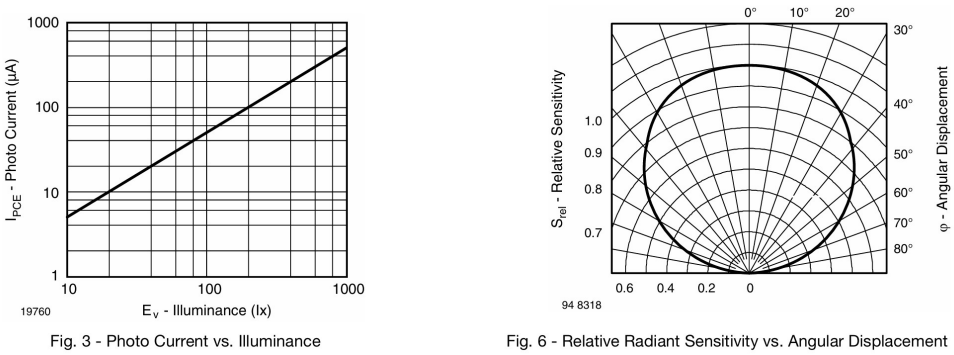

Abbildung 2.2: Auszug aus dem Datenblatt [1]

3 Lichtsteuerung

3.1 DIN-NORM

Zweck der Funktion:

- Automatisches Ein- und Ausschalten der Raumbeleuchtung basierend auf der gemessenen Beleuchtungsstärke im Raum durch Tageslicht.
- Vermeidung von zu häufigen Schaltvorgängen durch zeitliche Hysterese.
- Möglichkeit der manuellen Beeinflussung durch den Nutzer.

Realisierung:

- Nutzt Informationen von Präsenzerkennung, Belegungsstatus und Helligkeitsmessung.
- Übermittelt Schaltinformationen an den Lichtaktor.
- Optional: Manuelle Beeinflussung durch den Nutzer über die Bedienfunktion Licht stellen.

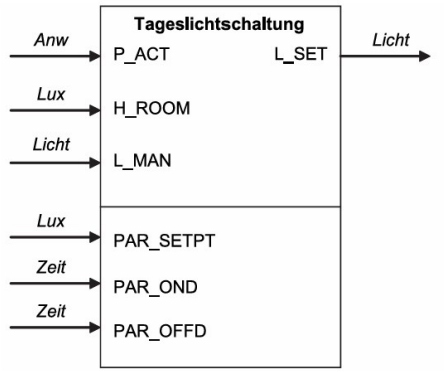

Abbildung 3.1: Funktionsblock der Tageslichsteuerung der DIN-Norm [2]

Abkürzungen im Funktionsblock:

Eingabeinformationen:

- **P_ACT:** Belegungszustand aus Belegungsauswertung/Präsenzerkennung.
- **H_ROOM:** Gemessene Beleuchtungsstärke am Arbeitsplatz.
- **L_MAN:** Übersteuerung durch den Nutzer.

Ausgabeinformationen:

- **L_SET:** Stellwert für zugehörige Aktorfunktionen (Lichtaktor).

Erforderliche Parameter:

- **PAR_SETPT:** Bestimmt die erforderliche Mindestbeleuchtungsstärke im Raum.
- **PAR_OND:** Einschaltverzögerung bei Änderung der Beleuchtungsstärke.
- **PAR_OFFD:** Abschaltverzögerung bei Änderung der Beleuchtungsstärke.

Beteiligte Ausgabeinformationen:

L_SET: Schaltet alle zugehörigen Aktorfunktionen des Lichtaktors, sowohl im Reglerbetrieb als auch bei Übersteuerung. [2]

3.2 UMSETZUNG DER NORM

- Der Eingabe von H_ROOM erfolgt über einen ADC-Pin. Jedoch wird nicht die Helligkeit im Raum gemessen sonder die Helligkeit außerhab des Hauses.
- Die Eingabe von P_ACT und L_MAN wird über den MQTT-Broker imititiert.
- Die Parameter PAR_SETPT, PAR_OND, PAR_OFFD werden ebenso über den MQTT-Broker gesetzt.
- Aufgrund von Problemen mit der Kompatibilität von Bibliotheken mit Circuitpython muss das Skript bezüglich der MQTT-Abfragen adaptiert werden. Die Beschreibungen der Adaption befinden sich in den Kommentaren des Codes: Lokal läuft das Skript auch auf dem Pico. Sollen Werte vom MQTT-Broker ausgelesen werden muss die Adaption nach den Kommentaren durchgeführt werden und ein Gerät, welches Python-fähig ist benutzt werden.

3.3 CODE

Die Erklärung für den Code befindet sich in den Kommentaren.

Listing 2: Lichtsteuerung

```
import board
import analogio
import digitalio
import mywifi
import mymqtt
import pwmio
import os
import time
from adafruit_datetime import datetime



#------------------- Definitions --------------------

# Connect to the MQTT Browser
mywifi.connect_wifi()
mqtt_client = mymqtt.connect_mqtt()

# LED
# use PWM to control the LED or use the digitalio to control
the LED

#led_pin = board.GP0       # Replace with the GPIO pin connected
to your LED
#PAR_SETPT = digitalio.DigitalInOut(led_pin)
#PAR_SETPT.direction = digitalio.Direction.OUTPUT

led1_pin = board.GP19       # Replace with the GPIO pin connected
to your LED
led1_pwm = pwmio.PWMOut(led1_pin)

# Create an AnalogIn object for the selected pin
ldr = analogio.AnalogIn(board.GP28)

# Configure the sun pin
sun_pin = board.GP18
sun_pwm = pwmio.PWMOut(sun_pin)



#------------------- MQTT --------------------

# Attention: Circuitpython does not have all the necessary libraries
to loop the check for changes in the variables on the MQTT server.
With local variables everything works.
#            The following code provides topics for the variables on
MQTT. The only thing we could not check is that if the variables change
on the MQTT server, the local variables will update.
#            There could be a problem with updating the variables.
Adjust the code as needed


# Local variables
```

```
PAR_SETPT = 300
L_MAN = False
P_ACT = False
PAR_OND = '08:00'
PAR_OFFD = '20:00'


# Conversion of the Sensor value to Lux
def lux(analog_value):
    return analog_value * 0.032 - 17.166

# Map a value from one range to another
def map_value(value, in_min, in_max, out_min, out_max):
    return (value - in_min) * (out_max - out_min) / (in_max - in_min)
    + out_min


# Function to control the brightness of the sun_pin
def sun():
    # Define the time it takes to transition from minimal to maximal
    brightness and vice versa
    transition_time = 60  # Adjust this value based on the desired
    transition time in seconds

    # Calculate the brightness based on the current time and transition
    time
    brightness = int(map_value(time.monotonic() % (2 * transition_time),
    0, transition_time * 2, 0, 65535))

    # Set the brightness of the sun_pin
    sun_pwm.duty_cycle = brightness


# Here we define a function that checks if the current time is in the
time slot
# We used adafruits datetime library to get the current time. For python
we would use the 'normal' datetime library, which has addictional functions
to get the current time. Adjust the code as needed
def is_current_time_in_timeslot(start_time_par, end_time_par):
    current_time = datetime.time()

    start_time = time(*map(int, start_time_par.split(':')))
    end_time = time(*map(int, end_time_par.split(':')))
    return start_time <= current_time <= end_time

start_time = PAR_OND
end_time = PAR_OFFD


# MQTT Topics
input_feed_P_ACT = "iot/Project_light_control/P_ACT"
input_feed_L_MAN = "iot/Project_light_control/L_MAN"
input_feed_PAR_OND = "iot/Project_light_control/PAR_OND"
input_feed_PAR_OFFD = "iot/Project_light_control/PAR_OFFD"

# Publishing MQTT topics
mqtt_client.publish("iot/Project_light_control/$Name"
```

```
, "Lichtsteuerung", retain=True)
mqtt_client.publish("iot/Project_light_control/$Ersteller", "Posch
,-Streicher,-Zischg", retain=True)
mqtt_client.publish("iot/Project_light_control/$state"
, "not-ready", retain=True)

mqtt_client.publish("iot/Project_light_control/P_ACT/$Name"
, "Praesenzerkennung", retain=True)
mqtt_client.publish("iot/Project_light_control/P_ACT/$Einheit"
, "None", retain=True)
mqtt_client.publish("iot/Project_light_control/P_ACT/$Datentyp"
, "bool", retain=True)

mqtt_client.publish("iot/Project_light_control/L_MAN/$Name"
, "Uebersteuerung", retain=True)
mqtt_client.publish("iot/Project_light_control/L_MAN/$Einheit"
, "None", retain=True)
mqtt_client.publish("iot/Project_light_control/L_MAN/$Datentyp"
, "bool", retain=True)

mqtt_client.publish("iot/Project_light_control/PAR_OND/$Name"
, "Startzeit", retain=True)
mqtt_client.publish("iot/Project_light_control/PAR_OND/$Einheit"
, "None", retain=True)
mqtt_client.publish("iot/Project_light_control/PAR_OND/$Datentyp"
, "int", retain=True)

mqtt_client.publish("iot/Project_light_control/PAR_OFFD/$Name"
, "Stopzeit", retain=True)
mqtt_client.publish("iot/Project_light_control/PAR_OFFD/$Einheit"
, "None", retain=True)
mqtt_client.publish("iot/Project_light_control/PAR_OFFD/$Datentyp"
, "int", retain=True)


def on_message(mqtt_client, topic, message):
    global PAR_OFFD_value
    global PAR_OND_value
    global P_ACT
    global L_MAN

    if topic == input_feed_P_ACT:
        # P_ACT button simulation
        print("Received-message-on-'P_ACT':", bool(message))
        P_ACT = bool(message)
    elif topic == input_feed_L_MAN:
        # L
        print("Received-message-on-'L_MAN':", bool(message))
        L_MAN = bool(message)
    elif topic == input_feed_PAR_OND:
        # PAR_OND value
        print("Received-message-on-'PAR_OND':", int(message))
        PAR_OND_value = int(message)
    elif topic == input_feed_PAR_OFFD:
        # PAR_OFFD value
        print("Received-message-on-'PAR_OFFD':", int(message))
        PAR_OFFD_value = int(message)
```

```
mqtt_client.on_message = on_message

mqtt_client.subscribe(input_feed_P_ACT, 0)
mqtt_client.subscribe(input_feed_L_MAN, 0)
mqtt_client.subscribe(input_feed_PAR_OND, 0)
mqtt_client.subscribe(input_feed_PAR_OFFD, 0)

mqtt_client.loop_start()

try:



    while True:

        #Sonnenfunktion funktioniert, aber der Sensor gibt komische
        Werte aus wenn die Sonne eingeschalten wird!
        #sun()

        # Get the values from the MQTT topics

        H_ROOM = lux(ldr.value)

        print("E-in-Lux", H_ROOM)

        # Update PAR_SETPT value based on values
        if (P_ACT and H_ROOM < PAR_SETPT) or L_MAN: # Turn on the
        light
            if is_current_time_in_timeslot(start_time, end_time):
                led1_pwm.duty_cycle = 2 ** 15
                # The print statement is only for debugging purposes
                and to check if the system is working
                print("Licht-ist-an")

# If the user stops the program, the program will stop the MQTT client
loop and the program will stop. We are not sure if this is the best
option for handeling the MQTT client. Adjust the code as needed
except KeyboardInterrupt:
    pass

mqtt_client.loop_stop()
```

4 Konstantlichtregelung

4.1 DIN-NORM

Zweck der Funktion:

Die Konstantlichtregelung regelt die Raumbeleuchtung oder Teile davon bei Belegung automatisch so, dass eine eingestellte Mindestbeleuchtungsstärke nicht unterschritten wird. Dadurch wird ein kontrastreiches Arbeiten bei gleichzeitig minimalem Energieeinsatz gewährleistet.

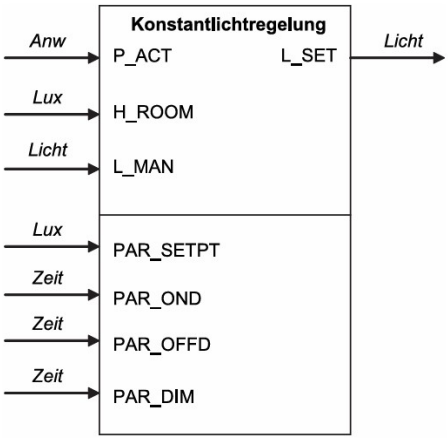

Abbildung 4.1: Funktionsblock der Konstantlichtregelung der DIN-Norm [2]

ABKÜRZUNGEN IM FUNKTIONSBLOCK:

Eingabeinformationen:

- **P_ACT:** Belegungszustand aus Belegungsauswertung/Präsenzerkennung.
- **H_ROOM:** Gemessene Beleuchtungsstärke am Arbeitsplatz.
- **L_MAN:** Übersteuerung durch den Nutzer.

Ausgabeinformationen:

- **L_SET:** (Regler-)Ausgangsgröße für zugehörige Aktorfunktionen (Lichtaktor).

ERFORDERLICHE PARAMETER:

- **PAR_SETPT:** Bestimmt die erforderliche Mindestbeleuchtungsstärke im Raum.
- **PAR_OND:** Einschaltverzögerung bei Änderung der Beleuchtungsstärke.
- **PAR_OFFD:** Abschaltverzögerung bei Änderung der Beleuchtungsstärke.
- **PAR_DIM:** Dimmrampe zur Anpassung der Änderung der Beleuchtungsstärke.

BETEILIGTE AUSGABEINFORMATIONEN:

L_SET: Über L_SET werden alle Aktorfunktionen des Lichtaktors, die der Konstantlichtregelung zugeordnet sind, sowohl im Reglerbetrieb als auch bei Übersteuerung gestellt. [2]

4.2 UMSETZUNG DER NORM

Die Konstantlichtregelung wird aufgrund hoher Komplexität auf den Demonstrationszweck getrimmt. Bei der normgerechten Umsetzung würde der Sensor normalerweise im zu steuernden Raum platziert werden. Im Modell wird der Photosensor aufgrund von Veranschaulichungszwecken auf dem Dach des Hauses installiert. Die Funktion, dass bei Abschattung das Licht heller wird und vice versa besteht jedoch.
Es ist wichtig zu beachten, dass diese Vorgehensweise von der Norm abweicht und lediglich zu Demonstrationszwecken dient.

4.3 CODE

- LED Ausgänge definieren
- Sonne mit Parameter definieren
- Initialisierung von MQTT-Komunikation und Analog-Eingänge
- Homie-Convention definieren
- Funktionen des Programms definieren
- Homie-Convention auf ready setzen
- Endlosschleife: Solange die Lux-Werte in einem definierten Bereich liegen, wird das Licht dementsprechend gedimmt. Sollten die Werte aus den Bereich treten, so erhalten die Lampen einen fixen Zustand (An / Aus). Solange der Sensor angesteckt ist, wird der Code ausgeführt. Bei Signalunterbrechung erfolgt ein ständiger Neustart, bis der Sensor wieder angeschlossen ist.

Listing 3: Konstantlichtregelung

```
import time
import board
import analogio
import digitalio
import pwmio
import mywifi
import mymqtt
import os
import microcontroller

#------------------ Definitions --------------------

analog_pin = board.GP28  # oder board.GP28

# Configure the room pins
led1_pin = board.GP21
led1_pwm = pwmio.PWMOut(led1_pin)
led2_pin = board.GP20
led2_pwm = pwmio.PWMOut(led2_pin)
led3_pin = board.GP19
led3_pwm = pwmio.PWMOut(led3_pin)

# Configure the sun pin
sun_pin = board.GP18
sun_pwm = pwmio.PWMOut(sun_pin)


#Connect to the MQTT Browser
mywifi.connect_wifi()
mqtt_client = mymqtt.connect_mqtt()

# Erstelle ein AnalogIn-Objekt f r den ausgew hlten Pin
analog_in = analogio.AnalogIn(analog_pin)

#Define a list of 100 values for the light density
light_densitiy_list = []
#Define a Publish rate
publish_rate = 0
publish_rate_max = 2000
# Set a Array Value
calc_light_const = 500

#------------------ MQTT --------------------

mqtt_client.publish("iot/Project_light_const/$Name"
, "Konstantlichsteuerung", retain = True)
mqtt_client.publish("iot/Project_light_const/$Ersteller"
, "Posch,-Streicher,-Zischg", retain = True)
mqtt_client.publish("iot/Project_light_const/$state"
, "not-ready", retain = True)
mqtt_client.publish("iot/Project_light_const/$$nodes"
, "Lichtintensit t,-Lichtkontrolle", retain = True)

output_feed_exception = "iot/Project_light_const/error"
output_feed_light_densitiy = "iot/Project_light_const
/light_densitiy"
output_feed_light_densitiy_brightness = "iot/Project_light_const
```

```
/light_densitiy/brightness"
output_feed_is_light = "iot/Project_light_const/is_light"

#------------------ Functions ------------------

# Define the calculation of the lights densitiy in lux
def lux(analog_value):
    return analog_value * 0.032 - 17.166

def get_light_density_levels():
    # Read the limits for the light density from the MQTT Browser
    # If the limits can not be read, set the default value to 1000
    light_densitiy_level_max = 600
    light_densitiy_level_min = 400

    return light_densitiy_level_max, light_densitiy_level_min


def light_density():
    # Read the light density from the sensor
    lux_value = lux(analog_in.value)
    # Add the new value to the list
    light_densitiy_list.append(lux_value)
    # If the list has more than 100 values, remove the first value
    if len(light_densitiy_list) > calc_light_const:
        light_densitiy_list.pop(0)
    # Calculate the average of the last 100 values
    light_densitiy = sum(light_densitiy_list) / len(light_densitiy_list)
    return light_densitiy

# Map a value from one range to another
def map_value(value, in_min, in_max, out_min, out_max):
    return (value - in_min) * (out_max - out_min)
    / (in_max - in_min) + out_min

# Set LED brightness based on light density
def set_led_brightness(lux_value, min_lux, max_lux):
    brightness = int(map_value(lux_value, min_lux, max_lux, 0, 65535))
    led1_pwm.duty_cycle = brightness
    led2_pwm.duty_cycle = brightness
    led3_pwm.duty_cycle = brightness


# Function to control the brightness of the sun_pin
def sun():
    # Define the time it takes to transition from minimal to maximal
    brightness and vice versa
    transition_time = 60  # Adjust this value based on the desired
    transition time in seconds

    # Calculate the brightness based on the current time and
    transition time
    brightness = int(map_value(time.monotonic() % (2 * transition_time)
    , 0, transition_time * 2, 0, 65535))

    # Set the brightness of the sun_pin
    sun_pwm.duty_cycle = brightness
```

```
#------------------ Homie-Setup ------------------

mqtt_client.publish("iot/Project_light_const/$state"
, "ready", retain = True)

mqtt_client.publish("iot/Project_light_const/light_densitiy/$Name"
, "Lichtintensitaet", retain = True)
mqtt_client.publish("iot/Project_light_const/light_densitiy/$Einheit"
, "Lux", retain = True)
mqtt_client.publish("iot/Project_light_const/light_densitiy/$Datentyp"
, "int", retain = True)

mqtt_client.publish("iot/Project_light_const/is_light/$Name"
, "Lichtkontrolle", retain = True)
mqtt_client.publish("iot/Project_light_const/is_light/$Einheit"
, "None", retain = True)
mqtt_client.publish("iot/Project_light_const/is_light/$Datentyp"
, "Bool", retain = True)

#------------------ Main ------------------
while True:


    # Read the limits for the light density from the MQTT Browser
    light_densitiy_level_max, light_densitiy_level_min
    = get_light_density_levels()

    try:
        # Check the light density
        lux_value = light_density()
        # Call the sun() function to control the brightness
        of the sun_pin
        sun()
        #If the light density is between the limits, turn the light ON
        if lux_value < light_densitiy_level_max and lux_value
        > light_densitiy_level_min:
            publish_rate += 1
            # If the light is not too bright, turn it ON.
            set_led_brightness(lux_value, light_densitiy_level_max
            , light_densitiy_level_min)
            # Publish the light status to the MQTT Browser
            if publish_rate == publish_rate_max:
                mqtt_client.publish(output_feed_is_light, "true"
                , retain = True)
                mqtt_client.publish(output_feed_light_densitiy, lux_value
                , retain = True)
                mqtt_client.publish(output_feed_light_densitiy_brightness
                , led1_pwm.duty_cycle, retain = True)
                publish_rate = 0
            time.sleep(0.001)
        elif lux_value < light_densitiy_level_min:
            publish_rate += 1
            # If the light is too bright, turn it constant ON.
            led1_pwm.duty_cycle = 65535
            led2_pwm.duty_cycle = 65535
            led3_pwm.duty_cycle = 65535
            # Publish the light status to the MQTT Browser
```

```
            if publish_rate == publish_rate_max:
                mqtt_client.publish(output_feed_is_light, "true"
                , retain = True)
                mqtt_client.publish(output_feed_light_densitiy, lux_value
                , retain = True)
                mqtt_client.publish(output_feed_light_densitiy_brightness
                , led1_pwm.duty_cycle, retain = True)
                publish_rate = 0
            time.sleep(0.001)
        else:
            publish_rate += 1
            # If the light is too bright, turn it OFF.
            led1_pwm.duty_cycle = 0
            led2_pwm.duty_cycle = 0
            led3_pwm.duty_cycle = 0
            # Publish the light status to the MQTT Browser
            if publish_rate == publish_rate_max:
                mqtt_client.publish(output_feed_is_light, "true"
                , retain = True)
                mqtt_client.publish(output_feed_light_densitiy, lux_value
                , retain = True)
                mqtt_client.publish(output_feed_light_densitiy_brightness
                , led1_pwm.duty_cycle, retain = True)
                publish_rate = 0
            time.sleep(0.001)


    except Exception as e:
        mqtt_client.publish(output_feed_exception, f"Error:-{e}"
        , retain = True)
        mqtt_client.publish("iot/Project_light_const/$state"
        , "not-ready", retain = True)
        print(f"Error:-{e}")
        time.sleep(3)
        #microcontroller.reset()
```

5 Modell

Folgendes Konzept wird angestrebt:

- Erscheinungsbild: EFH.
- LED simuliert Sonne.
- Sensor wird auf dem Dach montiert.
- Pico soll als Process Unit verbaut werden.
- Innenbeleuchtung: drei LEDs, welche mithilfe von PWM gedimmt werden können.
- Zur Besseren Visualisierung wird das Haus mit Spiegelfolie ausgelegt.

5.1 AUFBAU

5.1.1 CAD

Alle Bauteile wurden in Autodesk Inventor konstruiert. Folgende Abbildungen zeigen die 3D-Modelle der Bauteile:

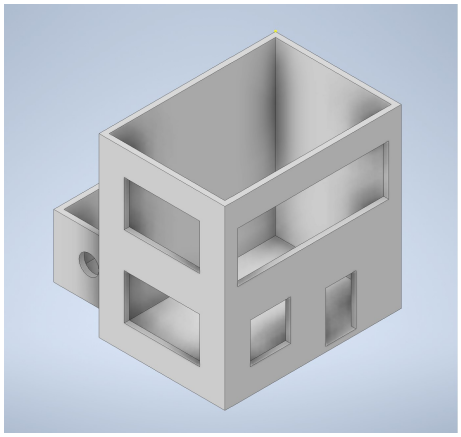

Abbildung 5.1: CAD-Modell des Hauses

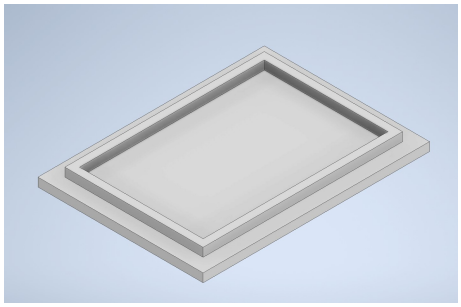

Abbildung 5.2: CAD-Modell des Hausdaches

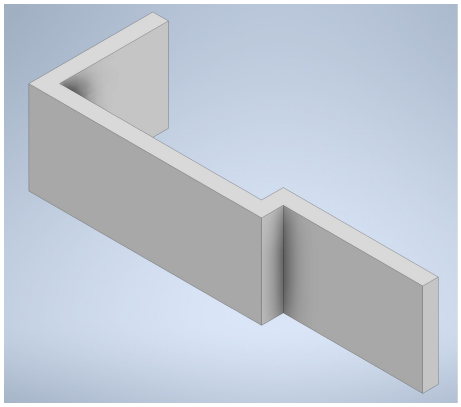

Abbildung 5.3: CAD-Modell der LED-Halterung

5.1.2 Assembly

Folgende Bilder zeigen das ausgedruckte und zusammengebaute Modell. Zu sehen ist der Photosensor am Dach des Hauses und die darüber montierte LED, welche die Sonne imitiert. An der Rückseite des Hauses ist ein Reservoir für den Pico.

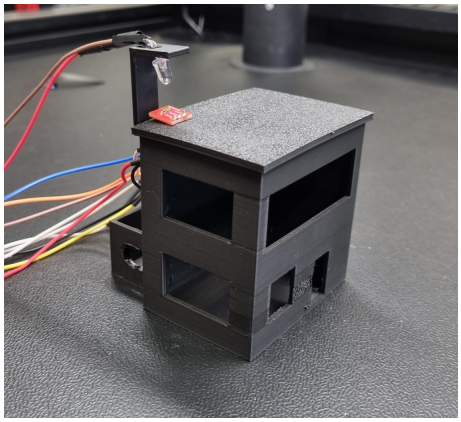

Abbildung 5.4: Gesamtansicht Haus

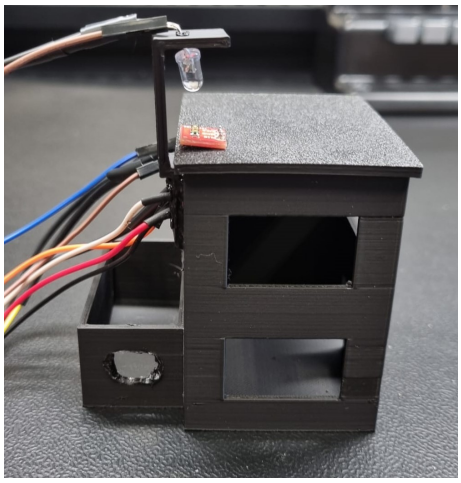

Abbildung 5.5: Seitenansicht Haus

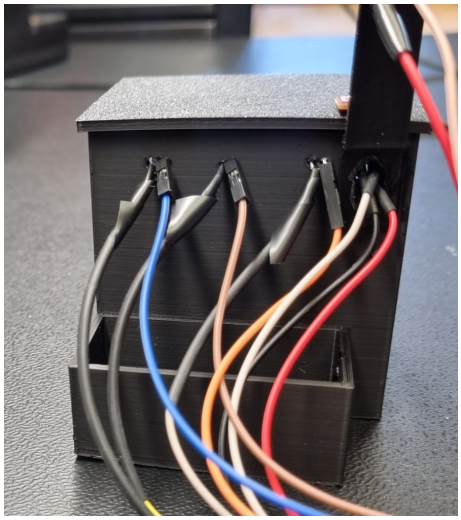

Abbildung 5.6: Rückseite Haus

6 Anleitung

6.1 BENÖTIGTE HARDWARE

- diverse LEDs
- Jumper Wires
- Strombegrenzende Widerstände für die LEDs
- RaspberryPi Pico
- TEMT6000 - Phtotsensormodul

6.2 ANSCHLUSS AN PICO

Folgende Pinbelegung für den Pico wird verwendet:

- LED1 = GP21
- LED2 = GP20
- LED3 = GP19
- Sonnen-LED = GP18
- Lichtsensor = GP28

In dieser Abbildung 6.1 ist das Pinout des Pico zu sehen:

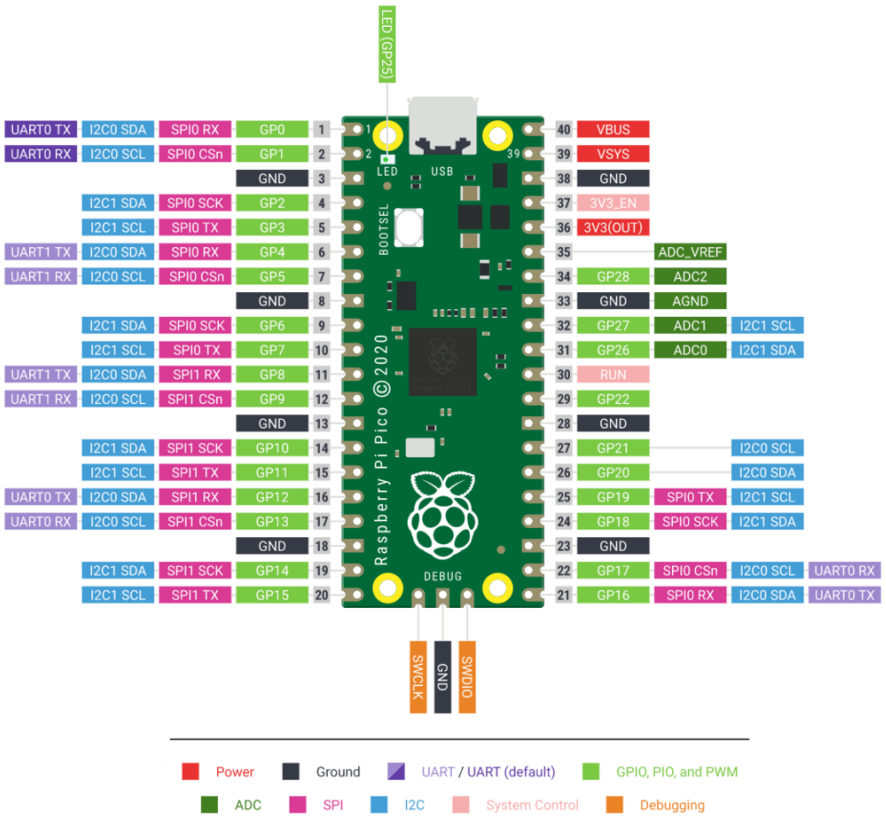

Abbildung 6.1: Pinout Raspberry Pi Pico WH

6.3 SETTINGS.TOML

Im .toml file werden die Anmeldedaten für die Verbindung des MQTT-Brokers und WLANs geschrieben.

6.4 INBETRIEBNAHME

- settings.toml wurde erfolgreich adaptiert.
- Komplette Ordnerstruktur inklusive .lib wird auf den Pico geflasht.
- Gewünschtes Python-Skript in code.py umbennen.
- Wird erneut Spannung an den Pico angelegt wird automatisch code.py ausgeführt.

6.5 ZUSATZ

Es scheint, als ob CircuitPython nicht alle erforderlichen Bibliotheken bereitstellt, um Änderungen in Variablen auf einem MQTT-Server zu überwachen. Der Code funktioniert gut mit lokalen Variablen, aber es gibt Schwierigkeiten, wenn sich die Werte auf dem MQTT-Server ändern und die lokalen Variablen nicht automatisch aktualisiert werden. Möglicherweise liegt dies an einem Problem bei der Aktualisierung der Variablen. Bitte überprüfen Sie den Code und passen Sie ihn an, um sicherzustellen, dass die Variablen ordnungsgemäß aktualisiert werden, wenn Änderungen auf dem MQTT-Server erkannt werden.

Abbildungsverzeichnis

Literaturverzeichnis

[1] Vishay Semiconductors, "TEMT6000," www.vishay.com, 2004.

[2] Verein deutscher Ingenieure, " Gebäudeautomation (GA) Raumautomationsfunktionen (RA-Funktionen)," VDI-Gesellschaft Bauen und Gebäudetechnik (GBG), 2011.