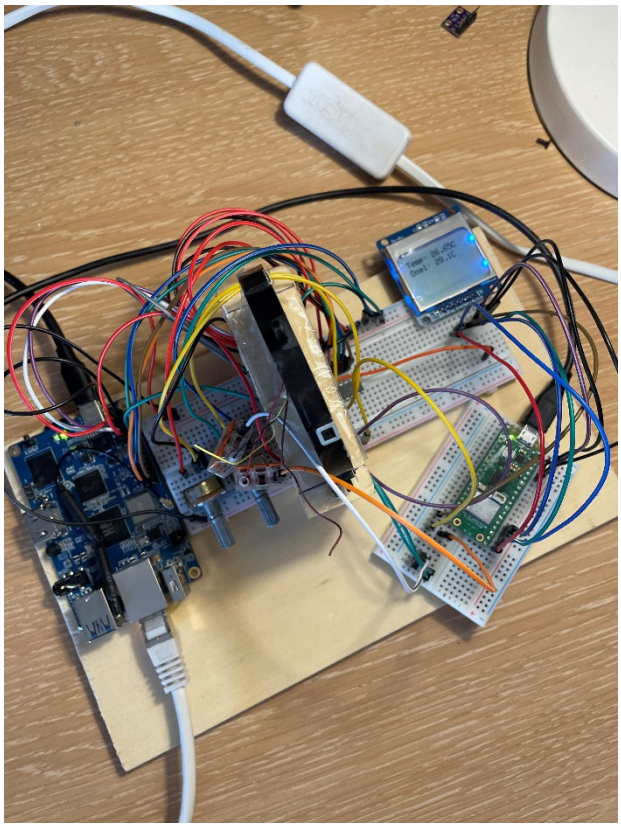
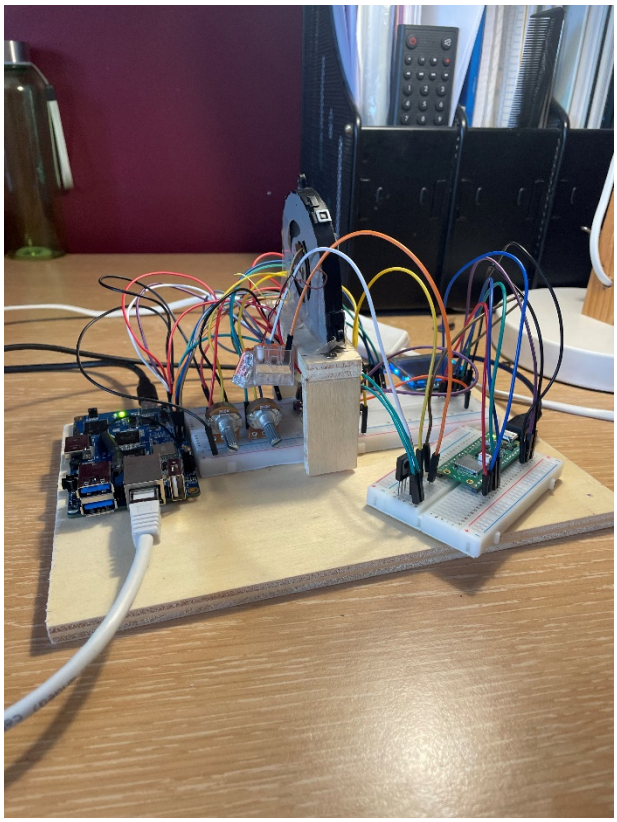
IOT Herkansing Examen

Roman Janssens





Thermostaat met Versneld Koelsysteem en Dubbele Potentiometers

Componenten:

Orangepi: Hoofdcontroller voor temperatuurmeting en regeling.

BMP280 sensor: Meet de huidige temperatuur en communiceert met de Orangepi.

Potentiometer 1: Stelt de gewenste (doel)temperatuur in via een ADC verbonden met de Orangepi.

Potentiometer 2: Regelt de snelheid van een ventilator om het koelingsproces te versnellen.

Weerstand: Verwarmingselement dat in thermisch contact staat met de BMP280 sensor.

Ventilator: Versnelt het koelingsproces om de temperatuur sneller naar de ingestelde waarde te brengen.

Transistor 1: Schakelt de stroom door de weerstand (verwarmingselement) om de temperatuur te verhogen.

Transistor 2: Regelt de werking van de ventilator om de temperatuur te verlagen.

Raspberry Pi Pico: Ontvangt de doel- en actuele temperatuur via MQTT en geeft deze weer op een LCD-scherm.

LCD-scherm: Toont de doel- en actuele temperatuur zoals ontvangen door de Raspberry Pi Pico.

MQTT: Protocol voor het versturen van gegevens tussen de Orangepi en de Raspberry Pi Pico.

Werking:

Temperatuurmeting: De BMP280 sensor meet continu de huidige temperatuur en stuurt deze naar de Orangepi.

Doeltemperatuur Instelling: Met de eerste potentiometer kan de gebruiker de gewenste temperatuur instellen. Deze waarde wordt door de Raspberry Pi gelezen via een ADC.

Temperatuurregeling:

De Orangepi vergelijkt de actuele temperatuur met de ingestelde doeltemperatuur.

Als de actuele temperatuur lager is dan de doeltemperatuur, verhoogt de Orangepi de stroom door de weerstand (via Transistor 1) om de temperatuur te verhogen.

Als je de gewenste temperatuur lager zet en je wil het sneller afkoelen kan je de tweede potentiometer gebruiken om snelheid van de fan met de 2de transistor.

Gegevensweergave:

De doel- en actuele temperatuur worden via MQTT naar een online dashboard gestuurd en daar weergegeven.

De Orangepi Pico ontvangt de doel- en actuele temperatuur via MQTT en toont deze op een LCD-scherm.

Dit systeem zorgt voor een efficiënte en automatische regeling van de temperatuur, waarbij zowel verwarming als koeling mogelijk is, afhankelijk van de ingestelde doeltemperatuur.

Self evaluation

Creativiteit: Ik denk dat ik een 1,5 op 2 verdien. Ik heb een ventilator toegevoegd voor snellere koeling en een extra potentiometer geïntegreerd, wat de functionaliteit heeft verbeterd.

Metingen: Ik denk dat ik een 3 op 3 verdien. Mijn metingen werken volledig zoals ze zouden moeten, zonder enige fouten. De temperatuur word juist gemeten

Potentiometer + ADC: Mijn potentiometer en ADC functioneren precies zoals bedoeld, en ik ben ervan overtuigd dat ze correct werken. Ik zou mezelf een 3 op 3 geven

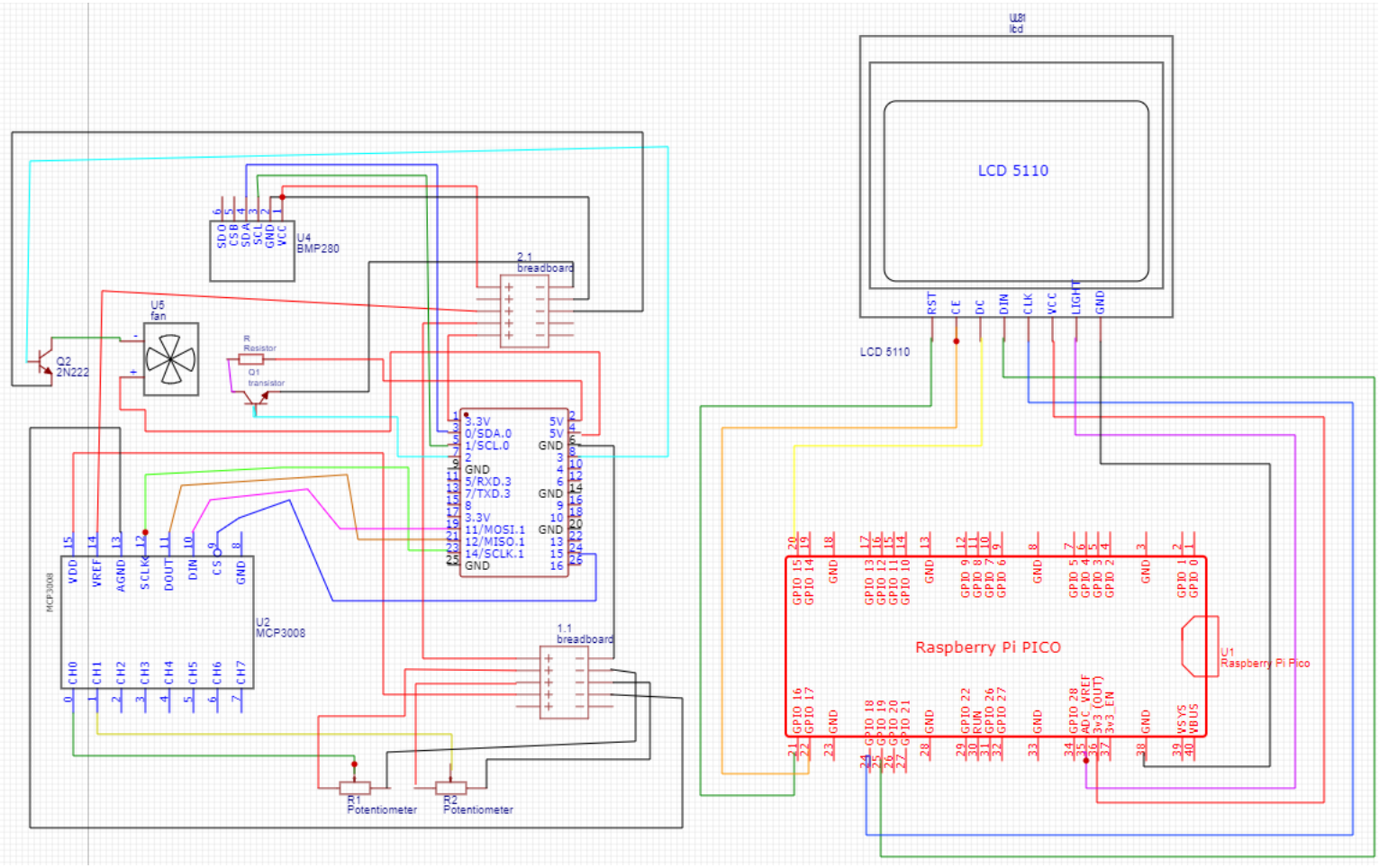
Automatische aanpassing: Ik zou hier een 2,5 op 3 geven. De temperatuur wordt automatisch op de gewenste waarde ingesteld dankzij de berekeningen die ik aan mijn code heb toegevoegd. Het PWM-signaal wordt berekend en omgezet om de doeltemperatuur te bereiken.

MQTT-dashboard (cloud): Mijn dashboard toont alle noodzakelijke informatie op ThingSpeak. De gegevens worden correct via MQTT verzonden. Ik zou denken een 3 op 3

MQTT-ontvangst (Pico): Ik heb een MQTT-broker gebruikt om de benodigde informatie naar de Raspberry Pico te sturen. Hier zou ik mezelf een 3 op 3 geven.

LCD (Pico): Ik vind dat ik een 3 op 3 verdien, omdat het LCD-scherm werkt zoals het hoort. De doeltemperatuur en de gemeten temperatuur worden correct weergegeven. Het scherm is verbonden met de Pico en niet enkel met de OrangePi.

Schematic


LCD 5110
RST
CE
DC
DIN
CLK
VCC
LIGHT
GND
Raspberry Pi PICO
U4
BMP280
U5
fan
Q2
2N222
R
Resistor
Q1
transistor
2.1
breadboard
1.1
breadboard
U2
MCP3008
R1
Potentiometer
R2
Potentiometer

Youtube Video

https://youtu.be/jKIu-qZb7kQ

RaspberryPico code :

```
import os
import wifi
import socketpool
import adafruit_minimqtt.adafruit_minimqtt as MQTT
import board
import busio
import digitalio
from adafruit_pcd8544 import PCD8544
import time

# Display-instellingen
spi = busio.SPI(clock=board.GP18, MOSI=board.GP19)
dc = digitalio.DigitalInOut(board.GP15)
cs = digitalio.DigitalInOut(board.GP17)
reset = digitalio.DigitalInOut(board.GP16)
display = PCD8544(spi, dc, cs, reset)
display.bias = 4
display.contrast = 60
display.invert = False
display.fill(0)
display.show()
```

```
# WiFi- en MQTT-instellingen
print("Verbinding maken met WiFi")
wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'), os.getenv('CIRCUITPY_WIFI_PASSWORD'))
print("Verbonden met WiFi")

pool = socketpool.SocketPool(wifi.radio)

mqtt_broker = "broker.hivemq.com"  # Publieke MQTT-broker
mqtt_port = 1883
mqtt_topic_temp = "orangepi/temperature"
mqtt_topic_goal = "orangepi/goal_temperature"

# Globale variabelen voor het opslaan van temperatuurwaarden
actual_temperature = None
goal_temperature = None

# Definieer callback-methoden
def connected(client, userdata, flags, rc):
    print("Verbonden met MQTT Broker!")

def disconnected(client, userdata, rc):
    print("Verbinding met MQTT Broker verbroken!")

def message(client, topic, message):
    global actual_temperature, goal_temperature

    if topic == mqtt_topic_temp:
        actual_temperature = message
        print(f"Ontvangen huidige temperatuur: {actual_temperature} °C")
    elif topic == mqtt_topic_goal:
```

```
        goal_temperature = message
        print(f"Ontvangen doeltemperatuur: {goal_temperature} °C")

    # Werk het display bij met de nieuwe temperaturen
    update_display()

def update_display():
    display.fill(0)
    if actual_temperature is not None:
        display.text(f"Temp: {actual_temperature}C", 0, 0, 1)
    if goal_temperature is not None:
        display.text(f"Doel: {goal_temperature}C", 0, 10, 1)
    display.show()

# Initialiseer de MQTT-client
mqtt_client = MQTT.MQTT(
    broker=mqtt_broker,
    port=mqtt_port,
    socket_pool=pool,
    keep_alive=60
)

# Stel de callback-methoden hierboven in
mqtt_client.on_connect = connected
mqtt_client.on_disconnect = disconnected
mqtt_client.on_message = message

# Maak verbinding met de MQTT-broker
print("Verbinding maken met MQTT-broker...")
mqtt_client.connect()
```

```
# Abonneer op de MQTT-onderwerpen
mqtt_client.subscribe(mqtt_topic_temp)
mqtt_client.subscribe(mqtt_topic_goal)
print("Geabonneerd op onderwerpen:", mqtt_topic_temp, mqtt_topic_goal)

# Continu controleren op berichten
while True:
    try:
        mqtt_client.loop(timeout=5)
    except Exception as e:
        print("Fout in loop:", e)
    time.sleep(1)
```

Orangepi code:

```
import time
import spidev
import wiringpi as GPIO
from smbus2 import SMBus
from bmp280 import BMP280
import requests
import paho.mqtt.client as mqtt

# ThingSpeak Configuratie
THINGSPEAK_API_KEY = "N4NN5TT3JG3D0PEP"
THINGSPEAK_URL = "https://api.thingspeak.com/update"
```

```
# MQTT Configuratie
mqtt_broker = "broker.hivemq.com"
mqtt_port = 1883
mqtt_topic_temp = "orangepi/temperature"
mqtt_topic_goal = "orangepi/goal_temperature"

# Initieer MQTT client
mqtt_client = mqtt.Client()
mqtt_client.connect(mqtt_broker, mqtt_port)

# Initieer I2C en BMP280
bus = SMBus(0)
bmp280 = BMP280(i2c_dev=bus)

# Initieer SPI voor ADC
SPI_PORT = 1
SPI_DEVICE = 0
spi = spidev.SpiDev()
spi.open(SPI_PORT, SPI_DEVICE)
spi.max_speed_hz = 1350000

# Initieer PWM voor eerste transistor
PWM_PIN_1 = 2
GPIO.wiringPiSetup()
GPIO.softPwmCreate(PWM_PIN_1, 0, 1024)
temp_min = 28.1
temp_max = 40

# Initieer PWM voor tweede transistor
PWM_PIN_2 = 3
```

```
GPIO.softPwmCreate(PWM_PIN_2, 0, 1024)



# Functie om ADC uit te lezen
def read_adc(channel):
    adc = spi.xfer2([1, (8 + channel) << 4, 0])
    data = ((adc[1] & 3) << 8) + adc[2]
    return data

# Eenvoudige moving average filter
def moving_average(new_value, values, N=5):
    values.append(new_value)
    if len(values) > N:
        values.pop(0)
    return sum(values) / len(values)

# Lege lijst voor ADC waarden
adc_values_1 = []
adc_values_2 = []

try:
    while True:
        # Lees en filter ADC waarde van kanaal 0 (eerste potentiometer)
        adc_value_1 = read_adc(0)
        filtered_adc_value_1 = moving_average(adc_value_1, adc_values_1)

        pwm_value_1 = int((filtered_adc_value_1 / 1023.0) * 1024)

        # Bereken de doeltemperatuur op basis van de omgekeerde PWM-waarde
        scale_factor = (temp_max - temp_min) / 1024
        goal_temp = temp_min + ((1024 - pwm_value_1) * scale_factor)
```

```
# Pas de omgekeerde PWM-waarde toe om het verwarmingselement te regelen
GPIO.softPwmWrite(PWM_PIN_1, pwm_value_1)



# Lees en filter ADC waarde van kanaal 1 (tweede potentiometer)
adc_value_2 = read_adc(1)
filtered_adc_value_2 = moving_average(adc_value_2, adc_values_2)

# Map en keer ADC waarde om naar PWM bereik voor de tweede transistor
pwm_value_2 = int((filtered_adc_value_2 / 1023.0) * 1024)
pwm_value_2 = 1024 - pwm_value_2

GPIO.softPwmWrite(PWM_PIN_2, pwm_value_2)

# Lees huidige temperatuur
actual_temp = bmp280.get_temperature()



# Toon temperatuur en PWM waarden in de terminal
print(f"Temperatuur: {round(actual_temp, 0)}°C, Doel Temp: {round(goal_temp,0)}")

# Stuur temperatuur en doeltemperatuur naar MQTT broker
try:
    mqtt_client.publish(mqtt_topic_temp, round(actual_temp,2))
    mqtt_client.publish(mqtt_topic_goal, round(goal_temp, 2))
except Exception as e:
    print(f"Fout bij verzenden naar MQTT broker: {e}")
```

```
        # Stuur gegevens naar ThingSpeak
        try:
            response = requests.post(THINGSPEAK_URL, params={
                'api_key': THINGSPEAK_API_KEY,
                'field1': actual_temp,
                'field2': round(goal_temp, 2),
            })

        except Exception as e:
            print(f"Fout bij verzenden naar ThingSpeak: {e}")

        time.sleep(0.1)

except KeyboardInterrupt:
    print("\nProgramma beëindigd")
```