**HUMIDITY SENSOR ALOGORITHM IN PYTHON**

import machine

import time

import dht

*# === a. Initialization ===*

*# Powering the sensor: assumed to be powered via Vcc/GND wired to STM32*

*# Configuration: Not needed for DHT22/DHT11 (they're simple digital sensors)*

*# Communication: Using single-wire digital protocol (handled by dht module)*

sensor\_pin = machine.Pin(4) # Replace with the actual GPIO pin used

humidity\_sensor = dht.DHT22(sensor\_pin) # Use DHT11 if applicable

*# === Optional Storage Buffer ===*

humidity\_log = []

def initialize\_sensor():

print("Initializing humidity sensor...")

*# No additional config needed for DHT22*

*# Communication is handled when the `measure()` function is called*

*return humidity\_sensor*

*# === b. Trigger measurement ===*

def trigger\_measurement(sensor):

try:

sensor.measure() # This triggers the DHT to take a new reading

except Exception as e:

print("Measurement trigger failed:", e)

*# === c. Read sensor Data ===*

def read\_humidity(sensor):

try:

humidity = sensor.humidity()

return humidity

except Exception as e:

print("Failed to read humidity:", e)

return None

*# === d. Data conversion ===*

*# DHT22 returns data in %RH directly — no raw ADC or conversion needed*

*# If using analog sensors, you'd read ADC and calibrate here*

*# === e. Data processing ===*

def process\_humidity(raw\_value):

# Simple filtering – moving average over last 3 values

if raw\_value is None:

return None

humidity\_log.append(raw\_value)

if len(humidity\_log) > 3:

humidity\_log.pop(0)

filtered = sum(humidity\_log) / len(humidity\_log)

return round(filtered, 2)

*# === f. Data storage and Transmission ===*

def store\_data(value):

*# Placeholder for logging or transmission*

print("Processed Humidity: {:.2f}%".format(value))

*# You can expand this to send to ThingSpeak, SD card, etc.*

*# === Main Loop ===*

sensor = initialize\_sensor()

while True:

trigger\_measurement(sensor)

raw\_humidity = read\_humidity(sensor)

processed\_humidity = process\_humidity(raw\_humidity)

if processed\_humidity is not None:

store\_data(processed\_humidity)

time.sleep(10) # Wait before next reading

**TEMPERATURE SENSOR ALGORITHM (E.G., DHT22 / DHT11)**

import machine

import time

import dht

*# === 1. Initialization ===*

sensor\_pin = machine.Pin(4) # Use your actual GPIO pin

temperature\_sensor = dht.DHT22(sensor\_pin)

*# === 2. Trigger Measurement ===*

def trigger\_temp\_measurement(sensor):

try:

sensor.measure()

except Exception as e:

print("Trigger failed:", e)

*# === 3. Read Sensor Data ===*

def read\_temperature(sensor):

try:

temp = sensor.temperature()

return temp

except Exception as e:

print("Read failed:", e)

return None

*# === 4. Data Conversion ===*

# DHT returns °C already — no need for conversion

*# === 5. Data Processing ===*

temp\_log = []

def process\_temperature(raw\_temp):

if raw\_temp is None:

return None

temp\_log.append(raw\_temp)

if len(temp\_log) > 3:

temp\_log.pop(0)

average\_temp = sum(temp\_log) / len(temp\_log)

return round(average\_temp, 2)

*# === 6. Data Storage ===*

def store\_temperature(value):

print("Processed Temperature: {:.2f} °C".format(value))

# You can store to memory/EEPROM/file here if needed

*# === Main Loop ===*

while True:

trigger\_temp\_measurement(temperature\_sensor)

temp\_raw = read\_temperature(temperature\_sensor)

temp\_processed = process\_temperature(temp\_raw)

if temp\_processed is not None:

store\_temperature(temp\_processed)

time.sleep(10)

**LIGHT INTENSITY SENSOR ALGORITHM (E.G., BH1750 OVER I2C)**

import machine

import time

import bh1750

*# === 1. Initialization ===*

i2c = machine.I2C(scl=machine.Pin(5), sda=machine.Pin(4))

light\_sensor = bh1750.BH1750(i2c)

*# === 2. Trigger Measurement ===*

# BH1750 starts measuring automatically upon reading

def trigger\_light\_measurement(sensor):

# No trigger needed for BH1750

pass

*# === 3. Read Sensor Data ===*

def read\_light\_intensity(sensor):

try:

lux = sensor.luminance(bh1750.BH1750.CONT\_HIRES\_1)

return lux

except Exception as e:

print("Light read error:", e)

return None

*# === 4. Data Conversion ===*

# BH1750 gives direct lux values — no conversion needed

*# === 5. Data Processing ===*

lux\_log = []

def process\_light(lux):

if lux is None:

return None

lux\_log.append(lux)

if len(lux\_log) > 5:

lux\_log.pop(0)

filtered\_lux = sum(lux\_log) / len(lux\_log)

return round(filtered\_lux, 2)

*# === 6. Data Storage ===*

def store\_light(value):

print("Processed Light Intensity: {:.2f} lx".format(value))

# You can store or log this value as needed

*# === Main Loop ===*

while True:

trigger\_light\_measurement(light\_sensor)

raw\_lux = read\_light\_intensity(light\_sensor)

processed\_lux = process\_light(raw\_lux)

if processed\_lux is not None:

store\_light(processed\_lux)

time.sleep(10).