**Thony code for Rasberry Pi Pico W to collect data from sensors and send it to thingspeak**

import network

import urequests

import utime

from machine import ADC, Pin

import dht

# === WiFi Credentials ===

SSID = "Yolo" # Replace with your WiFi SSID

PASSWORD = "12345678" # Replace with your WiFi Password

# === ThingSpeak API ===

THINGSPEAK\_API\_KEY = "4HLEXTX2B97YCNPY"

THINGSPEAK\_URL = "https://api.thingspeak.com/update"

# === Sensor Pins ===

adc\_voltage = ADC(26) # B25 Voltage Sensor (ADC0)

adc\_current = ADC(27) # ACS712 Current Sensor (ADC1)

dht\_sensor = dht.DHT11(Pin(16)) # DHT11 Temperature Sensor (GPIO16)

# === Connect to WiFi ===

def connect\_wifi():

wlan = network.WLAN(network.STA\_IF)

wlan.active(True)

wlan.connect(SSID, PASSWORD)

print("🔄 Connecting to WiFi...", end="")

while not wlan.isconnected():

utime.sleep(1)

print(".", end="")

print("\n✅ Connected to WiFi!")

print("IP Address:", wlan.ifconfig()[0])

# === Read Voltage (Convert ADC Value to Volts) ===

def read\_voltage():

raw\_value = adc\_voltage.read\_u16() # Read ADC (0-65535)

voltage = (raw\_value / 65535.0) \* 3.3 \* 5 # Convert to 0-25V range

return round(voltage, 2)

# === Read Current (Fixed Offset Calculation) ===

def read\_current():

raw\_value = adc\_current.read\_u16()

voltage = (raw\_value / 65535.0) \* 3.3 # Convert ADC value to voltage

# ✅ Auto-calibrate offset when no load is detected

measured\_offset = 0.2 # Measure this using a multimeter at no-load condition

sensitivity = 0.185 # ACS712-5A sensitivity = 185mV/A

current = (voltage - measured\_offset) / sensitivity # Convert voltage to current

print(f"Raw ADC: {raw\_value}, Voltage: {voltage:.2f}V, Current: {current:.2f}A")

return round(current, 2)

# === Read Temperature (Fixed) ===

def read\_temperature():

try:

utime.sleep(2) # ✅ DHT11 requires a delay before reading again

dht\_sensor.measure()

temperature = dht\_sensor.temperature()

return temperature

except OSError as e:

print("⚠ DHT11 Error: Sensor not responding.")

return -99 # Return a default error value

# === Send Data to ThingSpeak ===

def send\_to\_thingspeak(voltage, current, temperature):

if not network.WLAN(network.STA\_IF).isconnected():

print("⚠ WiFi Disconnected. Reconnecting...")

connect\_wifi()

try:

url = f"{THINGSPEAK\_URL}?api\_key={THINGSPEAK\_API\_KEY}&field1={current}&field2={voltage}&field3={temperature}"

response = urequests.get(url)

response.close()

print(f"✅ Data Sent: V={voltage}V, I={current}A, T={temperature}°C")

except Exception as e:

print("❌ Error Sending Data:", str(e))

# === Main Loop ===

def main():

connect\_wifi()

while True:

voltage = read\_voltage()

current = read\_current()

temperature = read\_temperature()

send\_to\_thingspeak(voltage, current, temperature)

utime.sleep(15) # Send data every 15 seconds

# Run the main function

main()

**Script for Matlab to train the ANN model to predict the SOC**

%% ===== STEP 1: Load Data from ThingSpeak =====

clc; clear; close all;

% Define your ThingSpeak channel ID and API key

channelID = 2877059; % Replace with your ThingSpeak Channel ID

apiKey = '4BDWVVUASBOMNZZW'; % Replace with your Read API Key (if private)

% Read data from ThingSpeak (last 800 samples)

numSamples = 700;

data = thingSpeakRead(channelID, 'NumPoints', numSamples, 'ReadKey', apiKey, 'OutputFormat', 'table');

% Check if data is empty

if isempty(data)

error('❌ Error: No data received from ThingSpeak! Check your API key and channel ID.');

end

% Display available column names for debugging

disp("✅ Available Columns in ThingSpeak Data:");

disp(data.Properties.VariableNames);

% Extract sensor readings (Check for correct field names)

voltage = data{:, 'voltage'};

current = data{:, 'current'};

% Detect temperature field name dynamically

if ismember('temperature', data.Properties.VariableNames)

temperature = data{:, 'temperature'};

elseif ismember('temprature', data.Properties.VariableNames)

temperature = data{:, 'temprature'};

else

error('❌ Error: Temperature field not found in ThingSpeak data. Available fields: %s', strjoin(data.Properties.VariableNames, ', '));

end

%% ===== STEP 2: Preprocess Data for ANN Training =====

% Normalize inputs to range [0,1]

Vmax = 12.6; % Set your max battery voltage

Vmin = 9.0; % Set your min battery voltage

SoC = (voltage - Vmin) / (Vmax - Vmin); % Normalize SoC between 0-1

% Prepare inputs and targets for training

inputs = [normalize(voltage)'; normalize(current)'; normalize(temperature)']; % Normalize inputs

targets = SoC'; % Target is SoC

% Check data dimensions

disp(size(inputs)); % Should be [3 x numSamples]

disp(size(targets)); % Should be [1 x numSamples]

%% ===== STEP 3: Train ANN Model =====

% Define a feedforward neural network with two hidden layers

net = feedforwardnet([10, 5]); % 10 neurons in layer 1, 5 in layer 2

% Set training parameters

net.trainParam.epochs = 500; % Number of training iterations

net.trainParam.goal = 1e-5; % Set error goal for accuracy

% Train the ANN

net = train(net, inputs, targets);

% Save the trained ANN model

save('ann\_soc\_model.mat', 'net', '-v7.3');

disp('✅ ANN training completed! Model saved as ann\_soc\_model.mat');

%% ===== STEP 4: Extract Weights and Export for Raspberry Pi Pico =====

% Load trained ANN

load('ann\_soc\_model.mat', 'net');

% Extract weights and biases

weights1 = net.IW{1,1}; % Input-to-Hidden Layer Weights

bias1 = net.b{1}; % Bias for Hidden Layer 1

weights2 = net.LW{2,1}; % Hidden-to-Output Layer Weights

bias2 = net.b{2}; % Bias for Output Layer

% Save in a format readable by MicroPython/C

save('ann\_parameters.mat', 'weights1', 'bias1', 'weights2', 'bias2');

disp('✅ ANN weights and biases exported successfully!');

% Write to a Python script for Raspberry Pi Pico

fileID = fopen('ann\_model.py', 'w');

fprintf(fileID, 'weights1 = %s\n', mat2str(weights1));

fprintf(fileID, 'bias1 = %s\n', mat2str(bias1));

fprintf(fileID, 'weights2 = %s\n', mat2str(weights2));

fprintf(fileID, 'bias2 = %s\n', mat2str(bias2));

fclose(fileID);

disp('✅ ANN model exported as ann\_model.py');

**Code for the Matlab function block which will call the saved script and find the SOC**

function soc = estimate\_soc(voltage, current, temperature)

%#codegen

persistent params;

% Load ANN Parameters only once

if isempty(params)

params = coder.load('ann\_parameters.mat'); % Load all data into a struct

end

% Extract ANN parameters

W1 = params.weights1;

b1 = params.bias1;

W2 = params.weights2;

b2 = params.bias2;

% === Normalize Inputs (Ensure Consistency with Training) ===

voltage\_min = 9.0; % Set according to training script

voltage\_max = 12.6;

current\_min = 0; % Adjust based on training data

current\_max = 5;

temp\_min = 20;

temp\_max = 50;

v\_norm = (voltage - voltage\_min) / (voltage\_max - voltage\_min);

i\_norm = (current - current\_min) / (current\_max - current\_min);

t\_norm = (temperature - temp\_min) / (temp\_max - temp\_min);

% === Forward Propagation (ANN Calculation) ===

x = [v\_norm; i\_norm; t\_norm]; % Input vector

% Hidden layer computation (ReLU activation)

hidden\_layer = max(0, W1 \* x + b1);

% Output layer computation (Linear activation)

soc = W2 \* hidden\_layer + b2;

% Ensure SoC is within valid range [0,1]

soc = min(max(soc, 0), 1);

end