Source code for the Intelligent System For Identifying Leaf Diseases In Soybean Crop

This code uses the ESP32-Cam camera to capture photos, the GY-NANO 6M GPS is responsible for purchasing date, time, latitude and longitude information that will be used for a ".csv" file, the LCD OLED 0.96" display. It has the function of informing the data that GPS is capturing to the user.

At the time of photo capture, the code generates a ".csv" list file that is associated with captured photo, gathering time and classification information (healthy, Asian rust, potassium deficiency, frog eye and target spot that this information They can be used in Power BI later.

The development environment used was the Arduino IDE.

```
// Libraries
#include "esp_camera.h"
#include "Arduino.h"
#include "FS.h"
#include "SD_MMC.h"
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
#include "driver/rtc_io.h"
#include <EEPROM.h>
#include <TinyGPS++.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Soja2024_v1_inferencing.h>  // Edge Impulse ML Inferencing System
#include "edge-impulse-sdk/dsp/image/image.hpp"

// Hardware Definition (camera configuration)
#define CAMERA_MODEL_AI_THINKER
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define I2C_SDA 15
#define I2C_SCL 14
TwoWire I2Cbus = TwoWire(0);
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C

/* Image Capture Definition ---------------------------------------------------- */
#define EI_CAMERA_RAW_FRAME_BUFFER_COLS 320
#define EI_CAMERA_RAW_FRAME_BUFFER_ROWS 240
#define EI_CAMERA_FRAME_BYTE_SIZE 3

// Variables for image manipulation and capture control
static bool debug_nn = false;
static bool is_initialised = false;
uint8_t *snapshot_buf; 102
```

```
// Camera Configuration
static camera_config_t camera_config = {

// ... (Camera PIN configuration)
.xclk_freq_hz = 20000000,
.ledc_timer = LEDC_TIMER_0,
.ledc_channel = LEDC_CHANNEL_0,
.pixel_format = PIXFORMAT_JPEG,
.frame_size = FRAMESIZE_QVGA,
.jpeg_quality = 12,
.fb_count = 1,
.fb_location = CAMERA_FB_IN_PSRAM,
.grab_mode = CAMERA_GRAB_WHEN_EMPTY,
};

// GPS and OLED Display configuration
static const int RXPin = 34, TXPin = 32;
static const uint32_t GPSBaud = 9600;
TinyGPSPlus gps;
HardwareSerial SerialGPS(1);
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

// Functions to boot/turn off the camera and capture the image
bool ei_camera_init(void);
void ei_camera_deinit(void);
bool ei_camera_capture(uint32_t img_width, uint32_t img_height, uint8_t *out_buf) ;
int pictureNumber = 0;
void setup() {
Serial.begin(115200);

// Camera Initialization
while (!Serial);
Serial.println("Edge Impulse Inferencing Demo");
if (ei_camera_init() == false) {
Serial.println("Failed to initialize Camera!");
} else {
Serial.println("Camera initialized");
}

// GPS Initialization
SerialGPS.begin(GPSBaud, SERIAL_8N1, RXPin, TXPin);

// OLED Display Initialization
if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
Serial.println(F("SSD1306 allocation failed"));
for(;;);
} 103
```

```
display.clearDisplay();
display.display();

// Load the photo number of EEPROM (non -volatile memory)
pictureNumber = EEPROM.read(0); // Reads the photo number stored in position 0 of EEPROM
EEPROM.write(0, pictureNumber + 1); // Increase the number for the next photo
EEPROM.commit(); // Save the photo number on EEPROM

// ... (SD card boot and other settings)
}
void loop() {

// --- GPS data reading ---
while (SerialGPS.available() > 0)
if (gps.encode(SerialGPS.read()))
if (gps.location.isValid())
break;

// --- OLED displya update ---
display.clearDisplay();
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,0);
display.print("Lat: "); display.println(gps.location.lat(), 6);
display.print("Lng: "); display.println(gps.location.lng(), 6);
display.display();

// Capturing the image (configure the function `ei_camera_capture`)
if (ei_camera_capture(EI_CLASSIFIER_INPUT_WIDTH, EI_CLASSIFIER_INPUT_HEIGHT,
snapshot_buf) == false) {
Serial.println("Failed to capture image\r\n");
return;
}

// Image classification (configure the function `run_classifier`)
ei::signal_t signal;
signal.total_length = EI_CLASSIFIER_INPUT_WIDTH * EI_CLASSIFIER_INPUT_HEIGHT;
signal.get_data = &ei_camera_get_data;
ei_impulse_result_t result = { 0 };
EI_IMPULSE_ERROR err = run_classifier(&signal, &result, debug_nn);
if (err != EI_IMPULSE_OK) {
ei_printf("ERR: Failed to run classifier (%d)\n", err);
return;
}

// OGetting the inference result with greater probability
float max_value = 0;
int max_index = 0;
for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
if (result.classification[ix].value > max_value) {
max_value = result.classification[ix].value; 104
```

```
max_index = ix;
}
}
const char* predicted_label = result.classification[max_index].label;

// Saving the image on the SD card (Create class folders in SD Card)
String filename = "/" + String(predicted_label) + "/" + String(pictureNumber) + ".jpg";
File file = SD_MMC.open(filename, FILE_WRITE);
if (!file) {
Serial.println("Failed to open file in writing mode");
return;
}
file.write(snapshot_buf, EI_CAMERA_RAW_FRAME_BUFFER_COLS *
EI_CAMERA_RAW_FRAME_BUFFER_ROWS * EI_CAMERA_FRAME_BYTE_SIZE);
file.close();

// --- Display "Foto Capturada" (captured photo)
display.fillRect(0, 48, SCREEN_WIDTH, 16, BLACK); // Limpa a parte inferior da tela
display.setCursor(0, 48);
display.print("Foto capturada!");
display.display();
delay(1000); // Exibe a mensagem por 1 segundo

// --- Creation/Update of the CSV file ---
String dataString = String(gps.location.lat(), 6) + ";" +
String(gps.location.lng(), 6) + ";" +
gps.date.value() + ";" +
gps.time.value() + ";" +
String(pictureNumber) + ";" +
String(predicted_label);
File csvFile = SD_MMC.open("/data.csv", FILE_APPEND);
if (csvFile) {
if (pictureNumber == 1) {
csvFile.println("latitude;longitude;data;hora;numero da foto;classificacao da foto");
}
csvFile.println(dataString);
csvFile.close();
} else {
Serial.println("Error opening data.csv");
}

// Increase the photo number for the next capture
pictureNumber++;

// Cleans the image capture buffer
free(snapshot_buf);
delay(1000); // Adjust this delay as needed
}
```