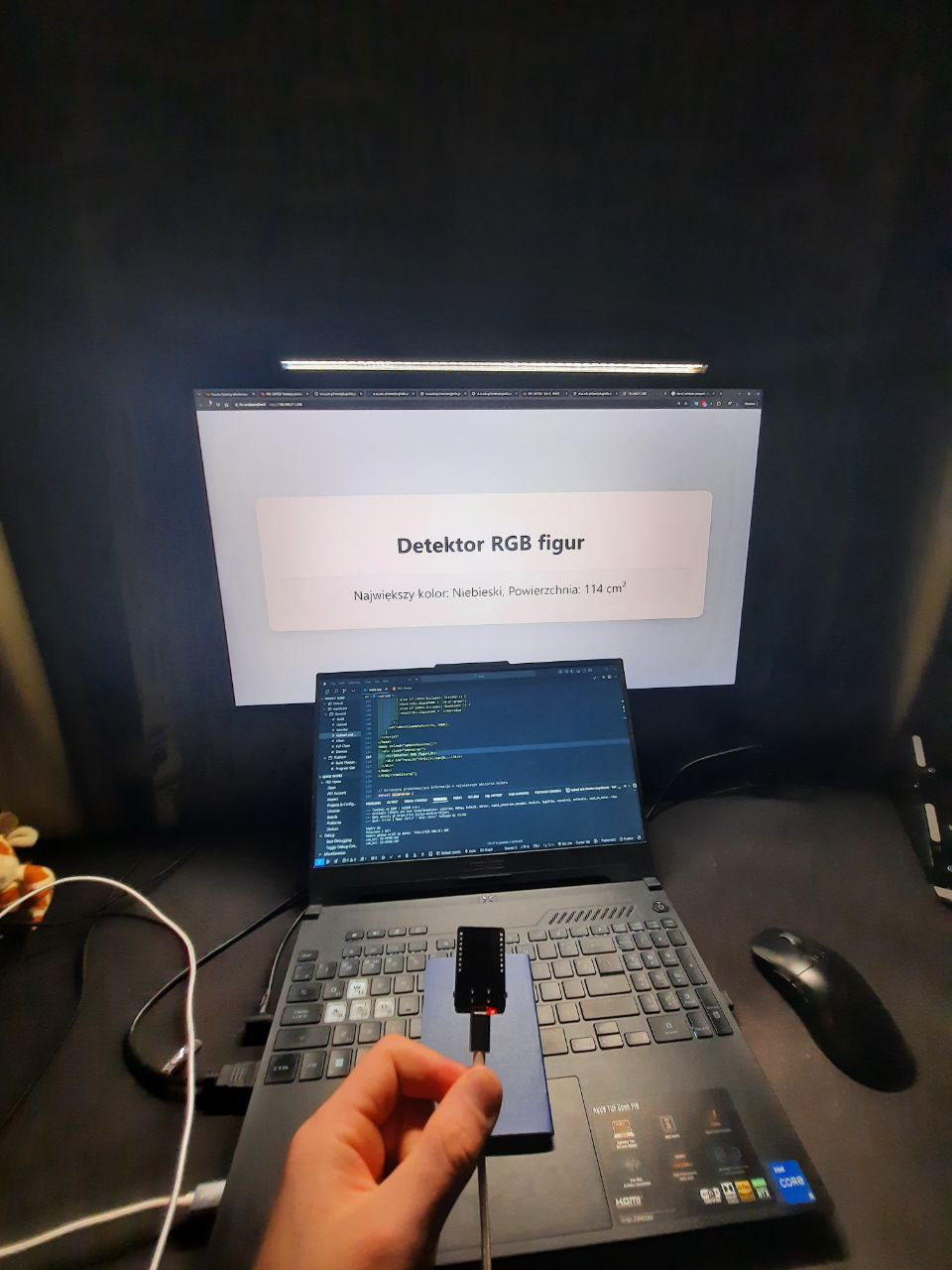
*Artem Zakharov, grupa A1*

*Systemy wbudowane*

*Zadanie 9*

***2024***



***Pole paszportu 110cm^2***

***Kod(zad-9):*** [***https://github.com/ZAKHAROV-Artem/systemy-wbudowane***](https://github.com/ZAKHAROV-Artem/systemy-wbudowane)

#include <WiFiClient.h>

#include <WiFi.h>

#include <ESPmDNS.h>

#include <esp\_camera.h>

#include <Arduino.h>

#include <esp\_timer.h>

#include <FS.h>

#include "ESPAsyncWebServer.h"

#define CAMERA\_MODEL\_AI\_THINKER

#define FRAME\_SIZE FRAMESIZE\_QQVGA

#define SOURCE\_WIDTH 160  // Szerokość źródłowego obrazu

#define SOURCE\_HEIGHT 120 // Wysokość źródłowego obrazu

#define BLOCK\_SIZE 5     // Rozmiar bloku do analizy (5x5 pikseli)

#define DEST\_WIDTH (SOURCE\_WIDTH / BLOCK\_SIZE)  // Szerokość po podziale na bloki

#define DEST\_HEIGHT (SOURCE\_HEIGHT / BLOCK\_SIZE) // Wysokość po podziale na bloki

// Zoptymalizowane parametry detekcji kolorów

const int PROG\_VALUE = 128;  // Próg detekcji koloru

const int OFFSET\_VALUE = 20; // Różnica między kolorami wymagana do klasyfikacji

const char\* PARAM\_INPUT\_1 = "input1";

const char\* PARAM\_INPUT\_2 = "offset";

String inputMessage;

int prog=128;

uint16\_t rgb\_frame[DEST\_HEIGHT][DEST\_WIDTH][3] = { 0 };

int offset=20;

#include "camera\_pins.h"

#include <SD.h>

#include <SPIFFS.h>

#define DIODA 33

#define CAMERA\_MODEL\_AI\_THINKER     //wybór modelu kamery

#include "camera\_pins.h"

const char\* ssid = "Сюда я";

const char\* password = "lalalala";

AsyncWebServer server(80);  //użycie serwera asynchronicznego http na porcie 80

//prosta strona www z miejscem na obraz z kamery

const char index\_html[] PROGMEM = R"rawliteral(

<!DOCTYPE HTML><html>

<head>

  <meta name="viewport" content="width=device-width, initial-scale=1">

  <style>

    body {

      font-family: 'Segoe UI', Arial, sans-serif;

      margin: 0;

      padding: 0;

      background: linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%);

      min-height: 100vh;

      display: flex;

      justify-content: center;

      align-items: center;

    }

    .container {

      width: 90%;

      max-width: 800px;

      background: white;

      padding: 30px;

      border-radius: 15px;

      box-shadow: 0 10px 20px rgba(0,0,0,0.1);

      margin: 20px;

    }

    h2 {

      color: #2c3e50;

      text-align: center;

      margin-bottom: 30px;

      font-size: 2.5em;

    }

    #results {

      background: #f8f9fa;

      padding: 20px;

      border-radius: 10px;

      font-size: 24px;

      color: #2c3e50;

      text-align: center;

      margin-top: 20px;

      border: none;

      box-shadow: inset 0 2px 4px rgba(0,0,0,0.05);

      transition: all 0.3s ease;

    }

    .color-red { color: #e74c3c; }

    .color-green { color: #2ecc71; }

    .color-blue { color: #3498db; }

  </style>

  <script>

    function updateResults() {

      fetch('/results')

        .then(response => response.text())

        .then(data => {

          const resultDiv = document.getElementById('results');

          resultDiv.innerHTML = data;

          // Dodanie kolorowania tekstu

          if (data.includes('Czerwony')) {

            resultDiv.className = 'color-red';

          } else if (data.includes('Zielony')) {

            resultDiv.className = 'color-green';

          } else if (data.includes('Niebieski')) {

            resultDiv.className = 'color-blue';

          }

        });

      setTimeout(updateResults, 2000);

    }

  </script>

</head>

<body onload="updateResults()">

  <div class="container">

    <h2>Detektor RGB figur</h2>

    <div id="results">Inicjalizacja...</div>

  </div>

</body>

</html>)rawliteral";

// Struktura przechowująca informacje o największym obszarze koloru

struct ColorArea {

    int size;        // Rozmiar obszaru w cm²

    String color;    // Nazwa koloru (Red, Green, Blue)

};

ColorArea currentLargestArea = {0, ""};

String lastResult = "";

// Funkcja przetwarzająca obraz z kamery na wartości RGB

void grab\_image(uint8\_t \*source, int len) {

     // Zerowanie tablicy RGB

     for (int y=0; y<DEST\_HEIGHT; y++) {

         for (int x=0; x<DEST\_WIDTH; x++) {

            rgb\_frame[y][x][0]=0; // Czerwony

            rgb\_frame[y][x][1]=0; // Zielony

            rgb\_frame[y][x][2]=0; // Niebieski

         }

     }

    // Przetwarzanie danych z kamery

    for (size\_t i = 0; i < len; i += 2) {

        // Konwersja danych z formatu RGB565 na RGB888

        const uint8\_t high = source[i];

        const uint8\_t low  = source[i+1];

        const uint16\_t pixel = (high << 8) | low;

        // Wyodrębnienie składowych RGB

        const uint8\_t r = (pixel & 0b1111100000000000) >> 11;

        const uint8\_t g = (pixel & 0b0000011111100000) >> 6;

        const uint8\_t b = (pixel & 0b0000000000011111);

        // Obliczenie pozycji w bloku

        const size\_t j = i / 2;

        const uint16\_t x = j % SOURCE\_WIDTH;

        const uint16\_t y = floor(j / SOURCE\_WIDTH);

        const uint8\_t block\_x = floor(x / BLOCK\_SIZE);

        const uint8\_t block\_y = floor(y / BLOCK\_SIZE);

        // Sumowanie wartości RGB dla każdego bloku

        rgb\_frame[block\_y][block\_x][0] += r;

        rgb\_frame[block\_y][block\_x][1] += g;

        rgb\_frame[block\_y][block\_x][2] += b;

    }

}

// Główna funkcja analizująca obraz

void fotka() {

    // Pobranie klatki z kamery

    camera\_fb\_t \* fb = NULL;

    fb = esp\_camera\_fb\_get();

    if (!fb) {

        Serial.println("Błąd przechwytywania obrazu");

        return;

    }

    grab\_image(fb->buf, fb->len);

    // Tablice do przechowywania informacji o kolorach

    char colorMap[DEST\_HEIGHT][DEST\_WIDTH];

    int redArea = 0, greenArea = 0, blueArea = 0;

    // Analiza kolorów w każdym bloku

    for (int y = 0; y < DEST\_HEIGHT; y++) {

        for (int x = 0; x < DEST\_WIDTH; x++) {

            // Sprawdzenie czy przeważa czerwony

            if ((rgb\_frame[y][x][0] > (rgb\_frame[y][x][1] + OFFSET\_VALUE)) &&

                (rgb\_frame[y][x][0] > (rgb\_frame[y][x][2] + OFFSET\_VALUE)) &&

                (rgb\_frame[y][x][0] > PROG\_VALUE)) {

                colorMap[y][x] = 'R';

                redArea++;

            }

            // Sprawdzenie czy przeważa zielony

            else if ((rgb\_frame[y][x][1] > (rgb\_frame[y][x][0] + OFFSET\_VALUE)) &&

                     (rgb\_frame[y][x][1] > (rgb\_frame[y][x][2] + OFFSET\_VALUE)) &&

                     (rgb\_frame[y][x][1] > PROG\_VALUE)) {

                colorMap[y][x] = 'G';

                greenArea++;

            }

            // Sprawdzenie czy przeważa niebieski

            else if ((rgb\_frame[y][x][2] > (rgb\_frame[y][x][0] + OFFSET\_VALUE)) &&

                     (rgb\_frame[y][x][2] > (rgb\_frame[y][x][1] + OFFSET\_VALUE)) &&

                     (rgb\_frame[y][x][2] > PROG\_VALUE)) {

                colorMap[y][x] = 'B';

                blueArea++;

            }

            else {

                colorMap[y][x] = ' ';

            }

        }

    }

    // Mnożnik do konwersji obszaru na cm²

    const float AREA\_MULTIPLIER = 0.3;

    // Znalezienie największego obszaru koloru

    if (redArea >= greenArea && redArea >= blueArea) {

        currentLargestArea = {(int)(redArea \* AREA\_MULTIPLIER), "Czerwony"};

    }

    else if (greenArea >= redArea && greenArea >= blueArea) {

        currentLargestArea = {(int)(greenArea \* AREA\_MULTIPLIER), "Zielony"};

    }

    else {

        currentLargestArea = {(int)(blueArea \* AREA\_MULTIPLIER), "Niebieski"};

    }

    // Aktualizacja wyniku

    lastResult = "Największy kolor: " + currentLargestArea.color +

                ", Powierzchnia: " + String(currentLargestArea.size) + " cm²";

    esp\_camera\_fb\_return(fb);

}

void setup() {

  Serial.begin(115200);

  Serial.setDebugOutput(true);

  Serial.println();

  camera\_config\_t config;

  config.ledc\_channel = LEDC\_CHANNEL\_0; //definicja portów, do których podłączona jes kamera

  config.ledc\_timer = LEDC\_TIMER\_0;

  config.pin\_d0 = Y2\_GPIO\_NUM;

  config.pin\_d1 = Y3\_GPIO\_NUM;

  config.pin\_d2 = Y4\_GPIO\_NUM;

  config.pin\_d3 = Y5\_GPIO\_NUM;

  config.pin\_d4 = Y6\_GPIO\_NUM;

  config.pin\_d5 = Y7\_GPIO\_NUM;

  config.pin\_d6 = Y8\_GPIO\_NUM;

  config.pin\_d7 = Y9\_GPIO\_NUM;

  config.pin\_xclk = XCLK\_GPIO\_NUM;

  config.pin\_pclk = PCLK\_GPIO\_NUM;

  config.pin\_vsync = VSYNC\_GPIO\_NUM;

  config.pin\_href = HREF\_GPIO\_NUM;

  config.pin\_sccb\_sda = SIOD\_GPIO\_NUM;

  config.pin\_sccb\_scl = SIOC\_GPIO\_NUM;

  config.pin\_pwdn = PWDN\_GPIO\_NUM;

  config.pin\_reset = RESET\_GPIO\_NUM;

  config.xclk\_freq\_hz = 20000000;

  config.pixel\_format = PIXFORMAT\_RGB565;

  config.frame\_size = FRAME\_SIZE;

  config.fb\_count = 1;

  esp\_err\_t err = esp\_camera\_init(&config);   //inicjacja kamery

  if (err != ESP\_OK) {

    Serial.printf("Błąd inicjacji kamery numer: 0x%x", err);

    return;

  }

  Serial.printf("kamera ok");

  sensor\_t \* s = esp\_camera\_sensor\_get();

  s->set\_framesize(s, FRAME\_SIZE);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL\_CONNECTED) {

    delay(500);

    Serial.print(".");

  }

  Serial.println("");

  Serial.println("Połączono z WIFI");

  Serial.print("Kamera gotowa wejdź na adres: 'http://");

  Serial.println(WiFi.localIP());

server.on("/", HTTP\_GET, [](AsyncWebServerRequest \*request){

    request->send(200, "text/html", index\_html);

  });

 server.on("/fotka", HTTP\_GET, [](AsyncWebServerRequest \*request){

  request->send(SPIFFS, "/photo.jpg", "image/jpg");

  });

  server.on("/results", HTTP\_GET, [](AsyncWebServerRequest \*request){

    request->send(200, "text/plain", lastResult);

  });

  server.begin();

}

void loop() {

  delay(2000);  // Opóźnienie 2 sekundy między pomiarami

  fotka();      // Wykonanie analizy obrazu

}