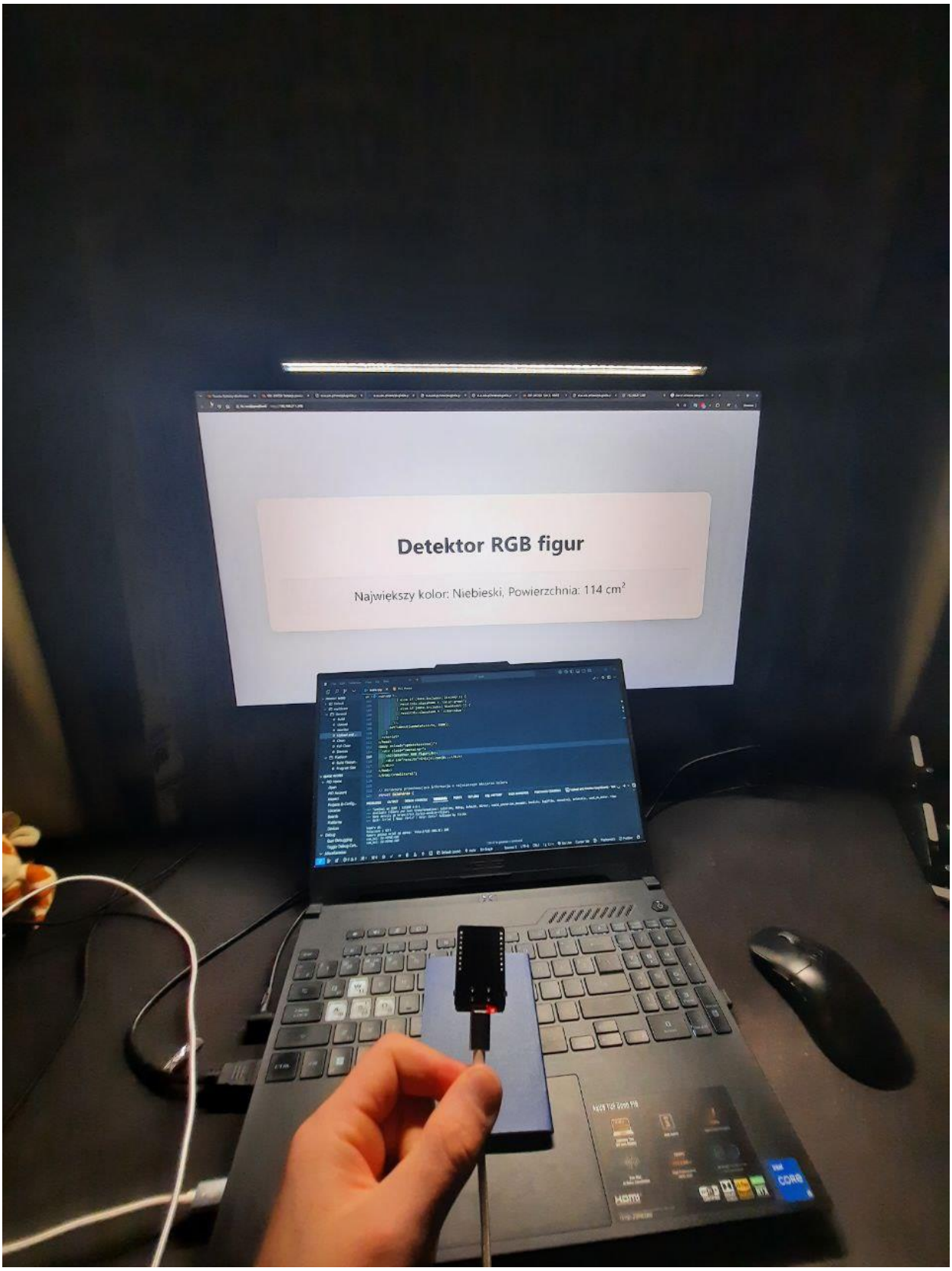
Artem Zakharov, grupa A1

Systemy wbudowane

Zadanie 9

2024


Detektor RGB figur
Największy kolor: Niebieski, Powierzchnia: 114 cm²

Pole paszportu 110cm^2

Kod(zad-9): 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
}
```