A black background with grey leaves

AI-generated content may be incorrect.

Iot color detection control board

Designed by abbas ghadiri

A grey logo on a black background

AI-generated content may be incorrect.

May 18, 2025

**ESP32 Color Detection System Documentation**

**Overview**

This project implements a color detection system using an ESP32 microcontroller with an OV2640 camera module. The system captures images, processes them to detect specific colors (brown, yellow, blue), determines their relative positions, and triggers actions (buzzer or relay) based on the detected positions. It also provides a web server to stream images and supports HTTP POST requests to send detection results to a specified endpoint.

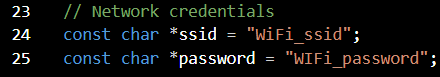
**Prerequisites**

* **Hardware**:
  + ESP32 module with OV2640 camera
  + LED (for camera flash, pin 4)
  + Status LED (pin 14)
  + Push button (pin 13)
  + Buzzer (pin 15)
  + Relay (pin 12)
* **Software**:
  + Arduino IDE or Platform-IO with ESP32 support
  + Libraries:
    - ESP\_camera (for camera control)
    - WiFi (for network connectivity)
    - WebServer (for HTTP server)
    - TickTwo (for timer-based callbacks)
    - HTTPClient (for HTTP POST requests)
    - ArduinoJson (for JSON serialization)
    - OpenCV (core, imgproc, imgcodecs for image processing)
* **Network**:
  + WiFi credentials (SSID and password)
  + Access to an NTP server (pool.ntp.org) for time synchronization
  + A server endpoint for HTTP POST requests (e.g., http://192.168.1.5/IoT\_color\_detection.php)

**Installation and Setup**

1. **Install Required Libraries**:  
   Install the necessary libraries via the Arduino Library Manager or Platform-IO:
2. **Configure Hardware**:
   * Connect the OV2640 camera to the ESP32 using the pin definitions in camera\_pins.h.
   * Connect the LED, status LED, push button, buzzer, and relay to the specified pins (4, 14, 13, 15, 12, respectively).
3. **Update Network Credentials**:

Modify the following constants in the code and replace with your WiFi SSID and password:



1. **Set Endpoint URL**:

Update the server endpoint for HTTP POST requests:



1. **Upload Code**:
   * Upload the code to the ESP32 using the Platform-IO.
   * Ensure the board is set to the correct ESP32 model (e.g., ESP32-CAM).
2. **Run the Program**:
   * Open the Serial Monitor (115200 baud) to view logs and the device's IP address.
   * Access the web server at http://<ESP32\_IP> to view the camera feed or http://<ESP32\_IP>/cd for cropped images.

**Code Structure**

* **Main File**: The primary source file contains all functionality.
* **Key Components**:
  + **Camera Configuration**: Initializes the OV2640 camera with QVGA resolution and RGB565 format.
  + **Color Detection**: Processes images to detect brown, yellow, and blue colors using OpenCV.
  + **Position Detection**: Calculates the center of detected color contours.
  + **Decision Logic**: Triggers a buzzer or relay based on the relative positions of detected colors.
  + **Web Server**: Serves full or cropped images via HTTP.
  + **Interrupt Handling**: Handles button presses with debouncing.
  + **HTTP POST**: Sends detection results (color positions and timestamp) to a server.
  + **NTP Client**: Synchronizes time for timestamping results.

**Key Functions**

* Position positionfinder(std::vector<std::vector<cv::Point>> contours): Calculates the center of contours for detected colors.
* void cropImage(uint8\_t \*input, int width, int height, int w, int h, const int w\_size, const int h\_size, uint8\_t \*output): Crops an image to a specified region of interest (ROI).
* void convertRGB565ToBGR(uint8\_t\* rgb565, int width, int height, cv::Mat& bgrImage): Converts RGB565 image data to BGR format for OpenCV processing.
* bool rangdetector(...): Determines if the yellow contour is between green and blue contours (not used in the main loop).
* String color\_detection(): Captures an image, detects colors, calculates positions, and triggers actions.
* void webpagehandler(unsigned char \*buffer, int length): Sends JPEG images to clients via the web server.
* void pagehandlermain(): Serves the full camera image.
* void pagehandler\_croped(): Serves a cropped image based on defined ROI.
* void http\_post(String timestamp, String colorcodes): Sends detection results to the server.

**Usage**

1. **Start the System**:
   * Power on the ESP32. It will connect to the specified WiFi network and initialize the camera.
   * The IP address will be printed to the Serial Monitor.
2. **Access the Web Server**:
   * Open a browser and navigate to http://<ESP32\_IP> to view the full camera feed.
   * Use http://<ESP32\_IP>/cd to view the cropped ROI image.
3. **Trigger Color Detection**:
   * Press the button (pin 13) to initiate color detection.
   * Alternatively, send the command r via the Serial Monitor to start detection.
   * Send s to stop detection or m to adjust the ROI parameters (x\_scale, y\_scale, x\_length, y\_length) via Serial input.
4. **View Results**:
   * The system detects brown, yellow, and blue colors in the ROI.
   * If the blue position is to the right of brown, and brown is to the right of yellow, the buzzer is activated; otherwise, the relay is triggered.
   * Results (color positions and timestamp) are sent to the specified endpoint via HTTP POST.
5. **Serial Commands**:
   * r: Run color detection.
   * s: Stop color detection.
   * m: Modify ROI parameters interactively.

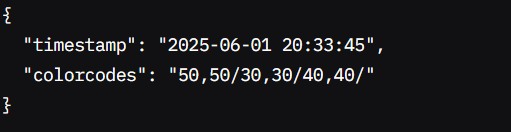
**Example Output**

* Serial Monitor (example):

A computer screen with white text

AI-generated content may be incorrect.

* HTTP POST Payload (example):



**Performance**

* **Camera**: Captures QVGA (320x240) images in RGB565 format.
* **Color Detection**: Uses HSV color space to detect brown, yellow, and blue colors with predefined ranges.
* **Position Analysis**: Computes the center of detected contours to determine relative positions.
* **Actions**:
  + Buzzer activates for 500ms if blue > brown > yellow in x-coordinates.
  + Relay activates for 500ms otherwise.
* **Web Server**: Streams JPEG images (full or cropped) with 80% quality.
* **Network**: Sends JSON payloads with detection results to the specified endpoint.

**Testing and Debugging**

* **Testing**:
  + Verify camera functionality by accessing the web server.
  + Test color detection by pressing the button or sending r via Serial.
  + Check HTTP POST responses in the Serial Monitor.
* **Common Issues**:
  + **Camera Failure**: Check pin connections and ensure camera\_pins.h matches your hardware.
  + **WiFi Disconnection**: Ensure correct SSID/password and stable network.
  + **Invalid ROI**: Use the m command to set appropriate ROI parameters.
  + **Memory Issues**: Ensure sufficient PSRAM for frame buffers.

**Limitations**

* Only detects brown, yellow, and blue colors with fixed HSV ranges.
* Limited to QVGA resolution due to ESP32 memory constraints.
* ROI parameters must be manually adjusted via Serial input.
* No real-time streaming for color detection results (only on button press or Serial command).

**Future Improvements**

* Add support for dynamic HSV range calibration.
* Implement real-time color detection without button presses.
* Enhance web server to display detection results.
* Add error handling for network failures.
* Support higher resolutions or additional color detection.

**Resources**

* [ESP32 Camera Library](https://github.com/espressif/esp32-camera)
* [OpenCV Documentation](https://docs.opencv.org/)
* [ArduinoJson Documentation](https://arduinojson.org/)
* [NTP Client Documentation](https://github.com/arduino-libraries/NTPClient)
* Full BOM [(Bill of Materials links to GitHub repository)](https://github.com/abbasghadir/IoT_color_detection/blob/main/hardware/IoT_color_detection_BOM.xlsx)
* Source code link [(GitHub repository)](https://github.com/abbasghadir/IoT_color_detection/tree/main/software)
* Schematic diagram and PCB layout [(GitHub repository)](https://github.com/abbasghadir/IoT_color_detection/tree/main/hardware)