Práctica: Detección de Colores mediante ESP32Cam y modelos de AI open source

Objetivo:

- Emplear un web-socket como medio de comunicación entre los datos ingresados a través de stream de video y un modelo pre-entredado de inteligencia Artificial
- Conocer y emplear modelos de inteligencia artificial con el objetivo de consumir API's en la nube

Introducción

Para poder desarrollar un sistema de detección de colores empleando el ESP32 es necesario conocer algunos conceptos importantes, que a continuación serán descritos a detalle:

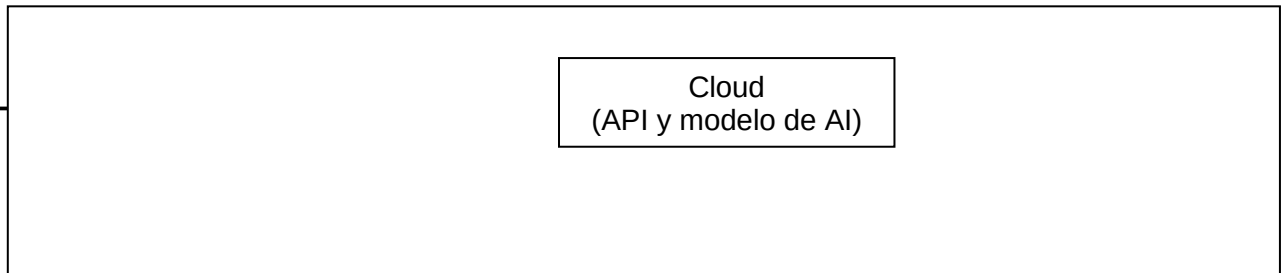
El primer concepto que debemos nuevamente contemplar es el de API que como bien sabemos es *interfaz de programación de aplicaciones* que en este caso tiene el objetivo concreto de integrar nuestro modelo de AI al sistema realizado con el SP32cam, por otro lado, también sabemos que para ello haremos uso de una técnica de asincronía web conocida como AJAX.

También es importante debido a que el objetivo de la práctica es más que nada el integrar AI en nuestro sistema embebido es por ello se partirá de un modelo previamente entrenado por Google, el cual llamaremos mediante javascript.

Por último, debido a que el proyecto ya contempla el uso de detección de colores haremos uso de uno de los modelos o representaciones de colores más comunes dentro del mundo web y programación que es el RGB, de manera general debemos contemplar que la combinación de Red, Green y Blue o colores primarios de este modelo nos darán como resultado toda la variedad o gama de colores a detectar.

Descripción:

En la presente práctica se abordará el uso de un web socket conectado a un API para poder llevar acabo la detección de objetos, para ello lo primero que debemos entender es un poco la arquitectura empleada en la presente práctica:



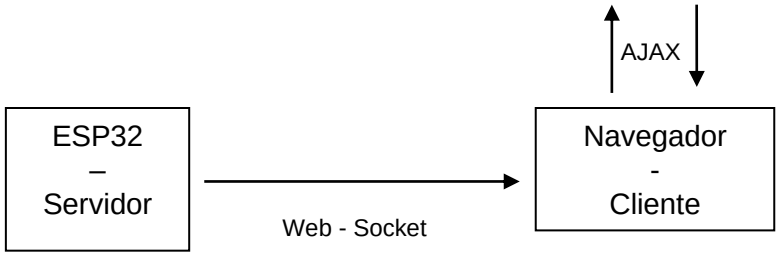


Imagen 1. Diagrama de componentes del sistema empleado

Podemos observar en la imagen 1 que lo primero que debemos hacer es el conectar el servidor con nuestro cliente mediante un web socket esto con el objetivo de transmitir el video adquirido mediante la cámara, además de que en el mismo servidor vamos a declarar las llamadas a la API a través del uso de AJAX.

Tabla de entradas y salidas:

Debido a que en este caso solamente se hará uso de la cámara de nuestro ESP32, no es necesario definir otras entradas y salidas, sin embargo, debemos saber que pines son los que se usan para la adquisición de datos a través de la cámara.

A continuación, se muestra la declaración de estos puertos dentro de nuestro código:

```
#define PWDN_GPIO_NUM    -1
#define RESET_GPIO_NUM   -1
#define XCLK_GPIO_NUM    21
#define SIOD_GPIO_NUM    26
#define SIOC_GPIO_NUM    27

#define Y9_GPIO_NUM      35
#define Y8_GPIO_NUM      34
#define Y7_GPIO_NUM      39
#define Y6_GPIO_NUM      36
#define Y5_GPIO_NUM      19
#define Y4_GPIO_NUM      18
#define Y3_GPIO_NUM       5
#define Y2_GPIO_NUM       4
#define VSYNC_GPIO_NUM   25
#define HREF_GPIO_NUM    23
#define PCLK_GPIO_NUM    22
```

Para más información acerca de los puertos dedicados exclusivamente para la cámara del ESP32, puede revisarse el datasheet del ESP32 - AI module o el del Wrove-module.

Diagrama de conexiones:

A continuación, se muestra el diagrama de conexiones de la presente práctica:

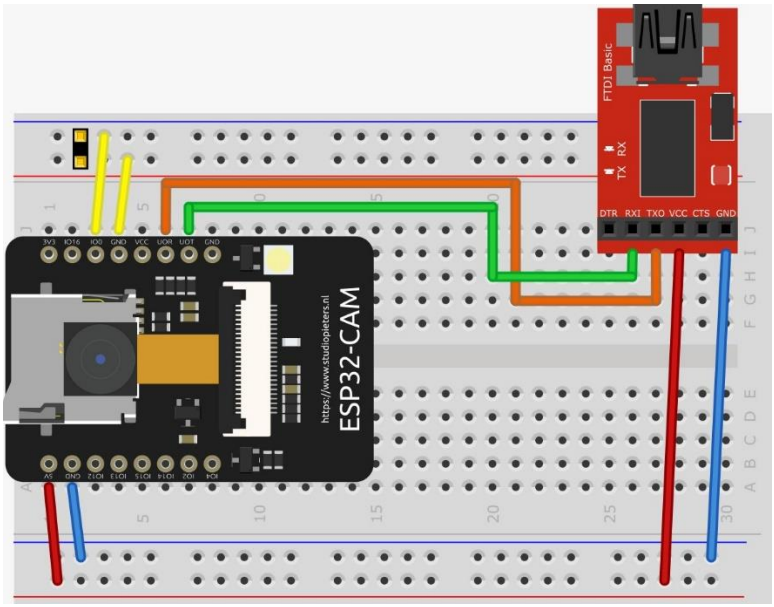


Imagen 2. Diagrama de conexiones en el ESP32

Recordemos que, una vez descargado el código en memoria, es necesario desconectar el pin 0 del 16.

Listado del programa y descripción:

En el caso particular de esta práctica se ha embebido el código tanto del cliente (HTMl , CSS y Javascript) además de los recursos para el web socket en un solo programa por lo que a continuación se describirán cada una delas funciones empeladas para el presente proyecto:

1. **Bibliotecas y variables globales**

En este caso lo más importante es recordar usar la biblioteca de la cámara del ESP32 además de incluir la biblioteca de wifi con la que conectaremos con otras como son soc y http_server que serán las encargadas de realizar el puente entre el cliente y servidor

```
2   const char* ssid     = "****";
3   const char* password = "***";
4
5   const char* apssid = "ESP32-CAM";
6   const char* appassword = "12345678";
7
8   #include <WiFi.h>
9   #include <WiFiClientSecure.h>
10  #include "esp_camera.h"
11  #include "soc/soc.h"
12  #include "soc/rtc_cntl_reg.h"
13
14  String Feedback="";
15  String Command="",cmd="",P1="",P2="",P3="",P4="",P5="",P6="",P7="",P8="",P9="";
16  byte ReceiveState=0,cmdState=1,strState=1,questionstate=0,equalstate=0,semicolonstate=0;
17
18
19  #define PWDN_GPIO_NUM     32
20  #define RESET_GPIO_NUM    -1
21  #define XCLK_GPIO_NUM      0
22  #define SIOD_GPIO_NUM     26
23  #define SIOC_GPIO_NUM     27
24
25  #define Y9_GPIO_NUM       35
26  #define Y8_GPIO_NUM       34
27  #define Y7_GPIO_NUM       39
28  #define Y6_GPIO_NUM       36
29  #define Y5_GPIO_NUM       21
30  #define Y4_GPIO_NUM       19
31  #define Y3_GPIO_NUM       18
32  #define Y2_GPIO_NUM        5
33  #define VSYNC_GPIO_NUM    25
34  #define HREF_GPIO_NUM     23
35  #define PCLK_GPIO_NUM     22
36
37  WiFiServer server(80);
```

Además y como parte del uso de nuestra cámara integrada, debemos hacer explícitos los pines que serán empleados, en este caso debido a que se decidió hacer lo más embebido posible el código solamente se importó parte del código de la biblioteca de cámara del ESP32 :

```
95    #define PWDN_GPIO_NUM     32
96    #define RESET_GPIO_NUM    -1
97    #define XCLK_GPIO_NUM      0
98    #define SIOD_GPIO_NUM     26
99    #define SIOC_GPIO_NUM     27
100
101   #define Y9_GPIO_NUM       35
102   #define Y8_GPIO_NUM       34
103   #define Y7_GPIO_NUM       39
104   #define Y6_GPIO_NUM       36
105   #define Y5_GPIO_NUM       21
106   #define Y4_GPIO_NUM       19
107   #define Y3_GPIO_NUM       18
108   #define Y2_GPIO_NUM        5
109   #define VSYNC_GPIO_NUM    25
110   #define HREF_GPIO_NUM     23
111   #define PCLK_GPIO_NUM     22
```

2. Funciones de stream y de websocket

Para el caso concreto de esta práctica se ha optimizado meramente el código dedicado a la detección de colores por lo que los filtros y codificaciones se han omitido parcialmente, en cambio en este caso todo se ha embebido en una sola función que tiene el objetivo de adquirir la imagen obtenida por el sensor:

```
60    else if (cmd=="resetwifi") {
61      WiFi.begin(P1.c_str(), P2.c_str());
62      Serial.print("Connecting to ");
63      Serial.println(P1);
64      long int StartTime=millis();
65      while (WiFi.status() != WL_CONNECTED)
66      {
67          delay(500);
68          if ((StartTime+5000) < millis()) break;
69      }
70      Serial.println("");
71      Feedback="STAIP: "+WiFi.localIP().toString();
72    }
73    else if (cmd=="restart") {
74      ESP.restart();
75    }
76    else if (cmd=="digitalwrite") {
77      ledcDetachPin(P1.toInt());
78      pinMode(P1.toInt(), OUTPUT);
79      digitalWrite(P1.toInt(), P2.toInt());
80    }
81    else if (cmd=="analogwrite") {
82      if (P1="4") {
83        ledcAttachPin(4, 4);
84        ledcSetup(4, 5000, 8);
85        ledcWrite(4,P2.toInt());
86      }
87      else {
88        ledcAttachPin(P1.toInt(), 5);
89        ledcSetup(5, 5000, 8);
90        ledcWrite(5,P2.toInt());
91      }
92    }
93    else if (cmd=="flash") {
94      ledcAttachPin(4, 4);
95      ledcSetup(4, 5000, 8);
```

3. **Página web y consumo de la API**

Para este apartado se empleó un arreglo de caracteres que tuviera la función de guardar todo el código HTMl y CSS de la página web además de incluir 2 principales scripts de JavaScript que son los encargados directamente de llamar el modelo de AI además de configurar su uso con los datos del stream de video.

3.1 Página web (HTML y CSS)

Para poder desarrollar una interfaz que sea visible mediante el navegador es necesario hacer explicito a nuestro web server el diseño mediante html y css, a continuación, se muestra parte del diseño de la interfaz:

```
<head>
<title>tracking.js - color with camera</title>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width,initial-scale=1">
<script src="https:\/\/ajax.googleapis.com/ajax/libs/jquery/1.8.0/jquery.min.js"></script>
<script src="https:\/\/fustyles.github.io/webduino/Tracking_20190917/tracking-min.js"></script>
</head><body>
<img id="ShowImage" src="" style="display:none">
<canvas id="canvas" width="0" height="0"></canvas><canvas id="canvas_custom" style="display:none"></canvas>
<table style="border:0px">
<tr style="border:0px">
  <td><input type="button" id="restart" value="Restart"></td>
  <td colspan="2"><input type="button" id="getStill" value="Start Detection">
  <!--<input type="checkbox" id="showpix" value="Show Pixel" onclick="if (this.checked) canvas_custom.style.display='block'; else c
  </td>
</tr>
<tr style="border:0px">
  <td style="text-align:center;background-color:red;"><a style="color:white;" onclick="changeTab('red');">Color 1 </a>
  <td style="text-align:center;background-color:green;"><a style="color:white" onclick="changeTab('green');">Color 2 </a></td>
  <td style="text-align:center;background-color:blue;"><a style="color:white" onclick="changeTab('blue');">Color 3 </a></td>
</td>
</tr><!---->
<tr>
  <td colspan="3">
  <div id="divColor1">
    <table style="border:4px red solid;width:100%;">
    <tr>
      <td align="right">R</td>
      <td colspan="2">min<input type="range" id="myColor_r_min1" min="0" max="255" value="0" step="1" onchange="myColor_r_min_v1.in
        max<input type="range" id="myColor_r_max1" min="0" max="255" value="0" step="1" onchange="myColor_r_max_v1.innerHTML=this.v
      </td>
    </tr>
    <tr>
      <td align="right">G</td>
      <td colspan="2">min<input type="range" id="myColor_g_min1" min="0" max="255" value="0" step="1" onchange="myColor_g_min_v1.in
        max<input type="range" id="myColor_g_max1" min="0" max="255" value="0" step="1" onchange="myColor_g_max_v1.innerHTML=this.v
      </td>
    </tr>
```

3.2 JavaScript

A diferencia de la práctica anterior en este caso a través de javascript es como haremos el llamado de nuevamente el API de Google de AJAX para el stream de video, sin embargo en este caso en vez de utilizar un modelo realizado por la misma Tensorflow (Google), haremos uso de un modelo open source de github que será cargado mediante el API de Google:

```
<script src="https:\/\/ajax.googleapis.com/ajax/libs/jquery/1.8.0/jquery.min.js"></script>
<script src="https:\/\/fustyles.github.io/webduino/Tracking_20190917/tracking-min.js"></script>
```

Por otro lado, tenemos un script embebido en el mismo html el cual tiene la función de hacer 2 funciones principal la primera es conectar el resultado arrojado por el API que es el color detectado y por otro el hacer el tracking o seguir el movimiento del color:

```
<script>
  var getStill = document.getElementById('getStill');
  var ShowImage = document.getElementById('ShowImage');
  var canvas = document.getElementById("canvas");
  var context = canvas.getContext("2d");
  var canvas_custom = document.getElementById('canvas_custom');
  var context_custom = canvas_custom.getContext('2d');
  var myColor = document.getElementById('myColor');
  var mirrorimage = document.getElementById("mirrorimage");
  var result = document.getElementById('result');
  var red = document.getElementById('red');
  var green = document.getElementById('green');
  var blue = document.getElementById('blue');
  var magenta = document.getElementById('magenta');
  var cyan = document.getElementById('cyan');
  var yellow = document.getElementById('yellow');
  var flash = document.getElementById('flash');
  var myTimer;
  var restartCount=0;
  var myColor_r_min1,myColor_r_max1,myColor_g_min1,myColor_g_max1,myColor_b_min1,myColor_b_max1;
  var myColor_r_min2,myColor_r_max2,myColor_g_min2,myColor_g_max2,myColor_b_min2,myColor_b_max2;
  var myColor_r_min3,myColor_r_max3,myColor_g_min3,myColor_g_max3,myColor_b_min3,myColor_b_max3;

  var tracker = new tracking.ColorTracker();

  tracking.ColorTracker.registerColor('red', function(r, g, b) {
    if ((r>=myColor_r_min1&&r<=myColor_r_max1)&&(g>=myColor_g_min1&&g<=myColor_g_max1)&&(b>=myColor_b_min1&&b<=myColor_b_max1)) {
      return true;
    }
    return false;
  });

  tracking.ColorTracker.registerColor('green', function(r, g, b) {
    if ((r>=myColor_r_min2&&r<=myColor_r_max2)&&(g>=myColor_g_min2&&g<=myColor_g_max2)&&(b>=myColor_b_min2&&b<=myColor_b_max2)) {
      return true;
    }
    return false;
  });

```

Además, en este mismo script es donde se realizan los cambios a nuestra página web, que a continuación se muestran algunos de estos cambios:

```
event.data.forEach(function(rect) {
  context.strokeStyle = rect.color;
  context.strokeRect(rect.x, rect.y, rect.width, rect.height);

  $.ajax({url:document.location.origin+'?serial='+rect.color+";"+rect.x+";"+rect.y+";"+rect.width+";"+rect.height+';stop',

  result.innerHTML+= rect.color+","+rect.x+","+rect.y+","+rect.width+","+rect.height+"<br>";
  if (rect.color=="red") {
    red.innerHTML+= rect.color+","+rect.x+","+rect.y+","+rect.width+","+rect.height+";";
  }
  else if (rect.color=="green") {
    green.innerHTML+= rect.color+","+rect.x+","+rect.y+","+rect.width+","+rect.height+";";
  }
  else if (rect.color=="blue") {
    blue.innerHTML+= rect.color+","+rect.x+","+rect.y+","+rect.width+","+rect.height+";";
  }
  else if (rect.color=="magenta") {
    magenta.innerHTML+= rect.color+","+rect.x+","+rect.y+","+rect.width+","+rect.height+";";
  }
  else if (rect.color=="cyan") {
    cyan.innerHTML+= rect.color+","+rect.x+","+rect.y+","+rect.width+","+rect.height+";";
  }
  else if (rect.color=="yellow") {
    yellow.innerHTML+= rect.color+","+rect.x+","+rect.y+","+rect.width+","+rect.height+";";
  }
});
});
```

Por último, a continuación, se muestra el resultado final de la interfaz e detección de colores:

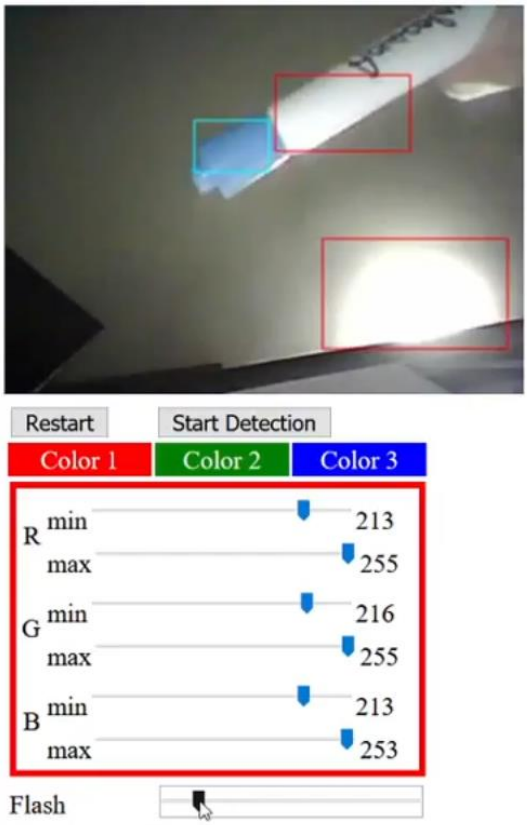


Imagen 3. Detección del color blanco y cyan mediante la cámara integrada en el ESp32cam