Práctica: Detección de Objetos mediante ESP32Cam y Tensorflow

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

El uso de sistemas embebidos como forma de adquisición de datos es una de las mayores tendencias dentro de las Ciencias de Datos, sin duda, el disponer de hardware con fines objetivos y sobre todo tan económicamente rentable de manufacturar y sustentar es lo que hace que sea posible destinar estas opciones a distintos mercados.

Por otro lado, y como bien se ha revisado en prácticas previas, el emplear sistemas web para la interacción y adquisición de datos nos da la versatilidad de poder disponer de nuestros datos al momento que nosotros queramos además de entrar en los denominados sistemas en tiempo real, que sin duda son además de llamativos una de las soluciones más convenientes en problemas de robótica.

En el caso de esta práctica, para poder desarrollar un sistema de detección de rostros empleando el ESP32 es necesario conocer algunos conceptos importantes, que a continuación serán descritos a detalle:

El primero es API, dentro del mundo del desarrollo comúnmente se habla de emplear o conectar sistemas a través del uso de API's, de manera general el término API es una abreviatura de **A**pplication **P**rogramming **I**nterfaces, que en español significa *interfaz de programación de aplicaciones*, es decir, es conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, sin embargo, el objetivo concreto de consumir una API en este proyecto es principalmente el ahorrarnos tiempo y esfuerzo al desarrollar, pues específicamente haremos uso de una de las API's más usadas en el desarrollo de Inteligencia Artificial que es Tensorflow además de emplear un modelo ya previamente entrenado.

Respecto a qué es TensorFlow es una biblioteca de código abierto para aprendizaje automático desarrollado por Google, específicamente en el caso de esta práctica emplearemos una versión web mediante el lenguaje de JavaScript esto con el objetivo de poder interactuar con nuestro web socket.

Además es importante debido a que el objetivo de la práctica es más que nada el integrar AI en nuestro sistema embebido es por ello que se partirá de un modelo previamente entrenado por Google , el cual se encuentra en un repositorio de Github del mismo TensorFlow https://github.com/tensorflow/tfjs-models/blob/master/coco-ssd/src/classes.ts.

Po r último, debido a que el proyecto ya contempla varias tecnologías como es el caso de consumir un modelo pre-entrenado, consumir una API web y conectar lo anterior en un web

socket, es por ello que debemos saber un poco acerca dela técnica web de desarrollo de este proyecto que es AJAX, por su acrónimo en inglés ***A**synchronous **J**avaScript **A**nd **X**ML* es una técnica meramente para desarrollo web asíncrono donde las aplicaciones se ejecutan en el cliente es decir en el navegador del consumidor de los desplegado por el servidor.

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:

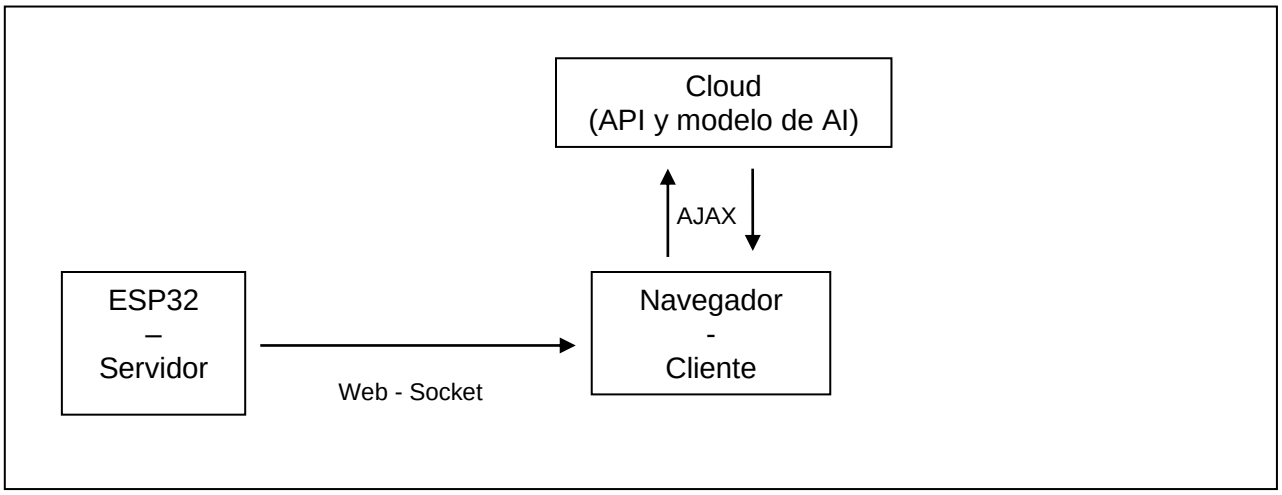

Imagen 1. Diagrama de componentes del sistema empleado

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:

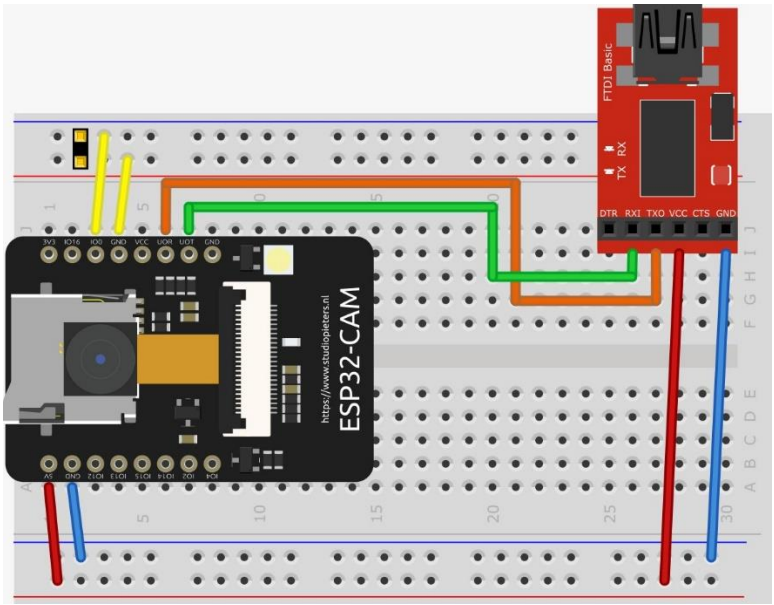

Imagen 2. Diagrama de conexiones en el ESP32

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     = "********";   //your network SSID
3   const char* password = "********";   //your network password
4
5   const char* apssid = "ESP32-CAM";
6   const char* appassword = "12345678";
7
8   #include <WiFi.h>
9   #include <esp32-hal-ledc.h>
10  #include "soc/soc.h"
11  #include "soc/rtc_cntl_reg.h"
12
13
14  #include "esp_camera.h"
15  #include "esp_http_server.h"
16  #include "esp_timer.h"
17  #include "img_converters.h"
18  #include "fb_gfx.h"
19  #include "fd_forward.h"
20
21  String Feedback="";
22  String Command="";
23  String cmd="";
24  String P1="";
25  String P2="";
26  String P3="";
27  String P4="";
28  String P5="";
29  String P6="";
30  String P7="";
31  String P8="";
32  String P9="";
33
34
35  byte ReceiveState=0;
36  byte cmdState=1;
37  byte strState=1;
38  byte questionstate=0;
39  byte equalstate=0;
40  byte semicolonstate=0;
```

Por otro lado, debemos definir algunas estructuras de datos destinadas al stream de video:

```
42  typedef struct {
43          size_t size; //number of values used for filtering
44          size_t index; //current value index
45          size_t count; //value count
46          int sum;
47          int * values; //array to be filled with values
48  } ra_filter_t;
49
50  typedef struct {
51          httpd_req_t *req;
52          size_t len;
53  } jpg_chunking_t;
54
55  #define PART_BOUNDARY "123456789000000000000987654321"
56  static const char* _STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;boundary=" PART_BOUNDARY;
57  static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
58  static const char* _STREAM_PART = "Content-Type: image/jpeg\r\nContent-Length: %u\r\n\r\n";
59
60  static ra_filter_t ra_filter;
61  httpd_handle_t stream_httpd = NULL;
62  httpd_handle_t camera_httpd = NULL;
```

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 :

```
92  // WARNING!!! Make sure that you have either selected ESP32 Wrover Module, ···
94
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

Con base en la práctica 4 y 5 sabemos de antemano que necesitamos hacer la conversión de los datos obtenidos de nuestro sensor de la cámara a datos binario para posteriormente pasarlos imágenes transportados y legibles en el stream de video, es por ello que a continuación se copiaron directamente esas funciones con el objetivo de solamente hacer uso de estas.

```
113 > void setup() {···
214
215 > void loop() {···
218
219 > static size_t jpg_encode_stream(void * arg, size_t index, const void* data, size_t len){···
230
231 > static esp_err_t capture_handler(httpd_req_t *req){···
300
301 > static esp_err_t stream_handler(httpd_req_t *req){···
411
412 > static esp_err_t cmd_handler(httpd_req_t *req){···
554
555
556 > static esp_err_t status_handler(httpd_req_t *req){···
575
```

Por otro lado, la parte diferenciadora respecto a las prácticas anteriores esla forma en que se ha configurado la función setup que es la encargada de hacer la instancia y llamado de objetos y funciones:

```
161   WiFi.mode(WIFI_AP_STA);
162   WiFi.begin(ssid, password);
163   delay(1000);
164   Serial.println("");
165   Serial.print("Connecting to ");
166   Serial.println(ssid);
167   long int StartTime=millis();
168   while (WiFi.status() != WL_CONNECTED) {
169     delay(500);
170     if ((StartTime+10000) < millis()) break;
171   }
172   if (WiFi.status() == WL_CONNECTED) {
173     WiFi.softAP((WiFi.localIP().toString()+"_"+(String)apssid).c_str(), appassword);
174     Serial.println("");
175     Serial.println("STAIP address: ");
176     Serial.println(WiFi.localIP());
177     for (int i=0;i<5;i++) {
178       ledcWrite(4,10);
179       delay(200);
180       ledcWrite(4,0);
181       delay(200);
182     }
183   }
184   else {
185     WiFi.softAP((WiFi.softAPIP().toString()+"_"+(String)apssid).c_str(), appassword);
186     for (int i=0;i<2;i++) {
187       ledcWrite(4,10);
188       delay(1000);
189       ledcWrite(4,0);
190       delay(1000);
191     }
192   }
193   startCameraServer();
194   pinMode(4, OUTPUT);
195   digitalWrite(4, LOW);
196 }
```

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:

```
<!doctype html>
<html>
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width,initial-scale=1">
        <meta http-equiv="Access-Control-Allow-Headers" content="Origin, X-Requested-With, Content-Type, Accept">
        <meta http-equiv="Access-Control-Allow-Methods" content="GET,POST,PUT,DELETE,OPTIONS">
        <meta http-equiv="Access-Control-Allow-Origin" content="*">
        <title> Object Detection</title>
        <style>
          body{font-family:Arial,Helvetica,sans-serif;background:■#181818;color:□#EFEFEF;font-size:16px}h2{font-size:18
        </style>
        <script src="https:\/\/ajax.googleapis.com/ajax/libs/jquery/1.8.0/jquery.min.js"></script>
        <script src="https:\/\/cdn.jsdelivr.net/npm/@tensorflow/tfjs@1.3.1/dist/tf.min.js"> </script>
        <script src="https:\/\/cdn.jsdelivr.net/npm/@tensorflow-models/coco-ssd@2.1.0"> </script>
    </head>
    <body>
        <section class="main">
            <section id="buttons">
                <table>
                <tr><td colspan="3"><canvas id="canvas" width="0" height="0"></canvas></td></tr>
                <tr><td><button id="restart" onclick="try{fetch(document.location.origin+'/control?restart');}catch(e){}"
                </table>
            </section>
            <figure>
              <div id="stream-container" class="image-container hidden">
                <div class="close" id="close-stream" style="display:none">×</div>
                <img id="stream" src="" style="display:none" crossorigin="anonymous">
              </div>
            </figure>
            <div id="logo">
                <label for="nav-toggle-cb" id="nav-toggle">&#9776;  Toggle settings</label>
            </div>
            <div id="content">
                <div id="sidebar">
                    <input type="checkbox" id="nav-toggle-cb">
                    <nav id="menu">
                        <div class="input-group" id="flash-group">
                            <label for="flash">Flash</label>
                            <div class="range-min">0</div>
                            <input type="range" id="flash" min="0" max="255" value="0" class="default-action">
```

3.2 JavaScript

Javascript es un lenguaje de programación orientado al desarrollo web, en el caso de esta práctica a través de principalmente 2 scripts embebidos y empleando 3 scripts externos (Montados en servidores externos o en la nube) es como podremos hacer posible la conexión con el modelo de inteligencia artificial de detección de objetos desarrollado por TensorFlow.

Como se puede apreciar en la imagen inferior, aquí podemos observar la importación de los scripts, 1 es para la conexión con el stream de video (AJAX), y los otros dos son los que hacen referencia al modelo de AI que está corriendo con NPM en servidores externos los cuales solamente los usaremos como una conexión a un API:

```
13      <script src="https:\/\/ajax.googleapis.com/ajax/libs/jquery/1.8.0/jquery.min.js"></script>
14      <script src="https:\/\/cdn.jsdelivr.net/npm/@tensorflow/tfjs@1.3.1/dist/tf.min.js"> </script>
15      <script src="https:\/\/cdn.jsdelivr.net/npm/@tensorflow-models/coco-ssd@2.1.0"> </script>
```

Por otro lado, los dos scripts embebidos en el ESP32 tienen 2 funciones principales, la primera es realizar los cambios dentro del HTML a través de eventos, y la segunda es hacer la conexión con los otros scripts externos, es decir hacer la carga del modelo de detección de objetos:

A continuación, parte del código encargado de hacer la interacción con la página web:

```
<script>
    document.addEventListener('DOMContentLoaded', function (event) {
      var baseHost = document.location.origin
      var streamUrl = baseHost + ':81'
      const hide = el => {
        el.classList.add('hidden')
      }
      const show = el => {
        el.classList.remove('hidden')
      }
      const disable = el => {
        el.classList.add('disabled')
        el.disabled = true
      }
      const enable = el => {
        el.classList.remove('disabled')
        el.disabled = false
      }
      const updateValue = (el, value, updateRemote) => {
        updateRemote = updateRemote == null ? true : updateRemote
        let initialValue
        if (el.type === 'checkbox') {
          initialValue = el.checked
          value = !!value
          el.checked = value
        } else {
          initialValue = el.value
          el.value = value
        }
        if (updateRemote && initialValue !== value) {
          updateConfig(el);
        }
      }
```

Por otro lado, a continuación, el script encargado de la carga y configuración del modelo de AI para la detección de objetos:

```
<script>
    var restart = document.getElementById('restart');
    var getStill = document.getElementById('get-still');
    var ShowImage = document.getElementById('stream');
    var canvas = document.getElementById("canvas");
    var context = canvas.getContext("2d");
    var result = document.getElementById('result');
    var Model;
    function LoadModel() {
      result.innerHTML = "Please wait for loading model.";
      cocoSsd.load().then(cocoSsd_Model => {
      Model = cocoSsd_Model;
      result.innerHTML = "";
      getStill.style.display = "block";
      getStill.click();
      });
    }
    function DetectImage() {
      canvas.setAttribute("width", ShowImage.width);
      canvas.setAttribute("height", ShowImage.height);
      context.drawImage(ShowImage, 0, 0, ShowImage.width, ShowImage.height);
      Model.detect(canvas).then(Predictions => {
      var s = (ShowImage.width>ShowImage.height)?ShowImage.width:ShowImage.height;
      if (Predictions.length>0) {
        result.innerHTML = "";
        for (var i=0;i<Predictions.length;i++) {
        const x = Predictions[i].bbox[0];
        const y = Predictions[i].bbox[1];
        const width = Predictions[i].bbox[2];
        const height = Predictions[i].bbox[3];
        context.lineWidth = Math.round(s/200);
        context.strokeStyle = "#00FFFF";
        context.beginPath();
        context.rect(x, y, width, height);
        context.stroke();
        context.lineWidth = "2";
        context.fillStyle = "red";
        context.font = Math.round(s/30) + "px Arial";
        context.fillText(Predictions[i].class, x, y);
        //context.fillText(i, x, y);
        result.innerHTML+= "[ "+i+" ] "+Predictions[i].class+", "+Math.round(Predictions[
        }

        for (var j=0;j<Predictions.length;j++) {
        if (Predictions[j].class=="person"&&Predictions[j].score>=0.5) {
          try{
            $.ajax({url: document.location.origin+'/control?serial='+Predictions[j].cla
          break;
          }
          catch(e){}
        }
        }
      }
      else
        result.innerHTML = "Unrecognizable";
      getStill.click();
      });
    }
    ShowImage.onload = function (event) {
      if (Model) {
      try {
        document.createEvent("TouchEvent");
        setTimeout(function(){DetectImage();},250);
      }
      catch(e) {
        DetectImage();
      }
      }
    }
    window.onload = function () { LoadModel(); }
    </script>
    </body>
</html>)rawliteral";
```

Como apartado final de esta práctica se muestra una captura general de la interfaz en funcionamiento:

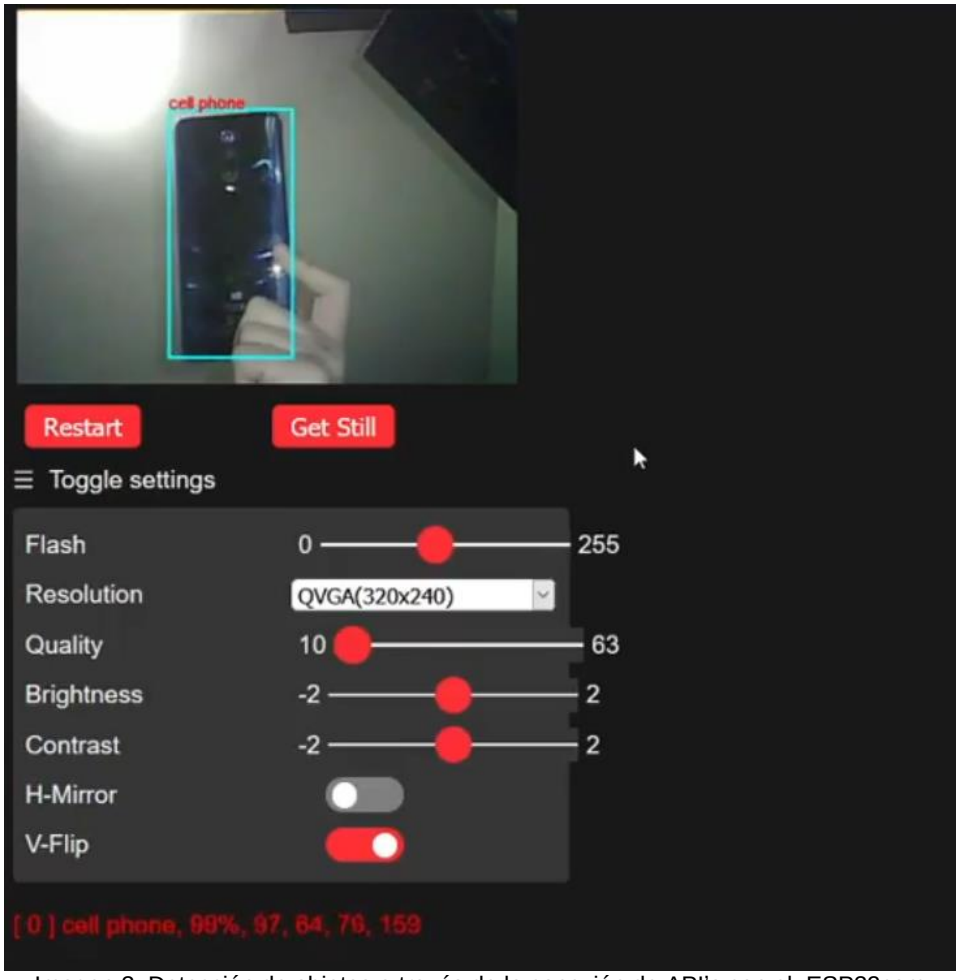

Imagen 3. Detección de objetos a través de la conexión de API's con el ESP32cam