

**Universidad Nacional Autónoma de México**

**Facultad de Ingeniería**

**Laboratorio de Microcomputadoras**

**Proyecto Final: Detección de objetos en microcontrolador  
ESP32**

Profesor:

- Rubén Anaya García

Alumnos:

- Murrieta Villegas Alfonso
- Reza Chavarría, Sergio Gabriel
- Valdespino Mendieta Joaquín

Grupo: 01

Semestre: 2021-2

# Proyecto Final: Detección de objetos en microcontrolador ESP32

## Objetivo

- Emplear los conceptos aprendidos en la materia de Microcomputadoras con la finalidad de crear un proyecto llamativo
- Utilizar un web-socket como medio de comunicación entre los datos ingresados a través de stream de video y un modelo pre-entrenado de Inteligencia Artificial
- Consumir los recursos disponibles de un microcontrolador con el objetivo de crear un sistema de bajo costo, pero con amplias posibilidades de empleabilidad

## Introducción

Una de las mayores tendencias en la industria 4.0 es el concepto de “*Internet of Things*” que se caracteriza por las amplias posibilidades que se puede tener o crear con componentes de bajo precio en distintos campos, ejemplos como el hacer más cómodas las rutinas humanas hasta aspectos específicos del mundo industrial como la agrociencia son resultado de esta tendencia.

Por otro lado, 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*.

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

## Marco Teórico

### 1. *Conceptos de Software*

El término API (*Application Programming Interfaces*) o interfaz de programación de aplicaciones, es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar *software de las aplicaciones*, es decir, el objetivo concreto de consumir una API en nuestro proyecto es principalmente el ahorrarnos el desarrollo en componentes

que otros desarrolladores han realizado, específicamente haremos uso de una de las API's más usadas en el desarrollo de Inteligencia Artificial que es la de Tensorflow, además de emplear un modelo de Inteligencia Artificial previamente entrenado.

El modelo entrenado por Google, se encuentra en el siguiente repositorio de Github <https://github.com/tensorflow/tfjs-models/blob/master/coco-ssd/src/classes.ts>.

Por 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, debemos a su vez considerar la técnica web de desarrollo de este proyecto, la cual es AJAX.

Por su acrónimo en inglés *Asynchronous JavaScript And XML* 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.

## ***2. Conceptos de Hardware***

Lo primero que debemos saber o conocer respecto al hardware empleado en el presente proyecto es nuestro microcontrolador, que en este caso es el desarrollador por la empresa china *Expressif*, **ESP32 versión CAM**, este microcontrolador en específico y como bien lo menciona en el mismo nombre se caracteriza por integrar una cámara además de un pequeño flash.



*Imagen 1. Microcontralor ESP32-CAM*

Como se puede observar en la imagen anterior, además de estar destinado a la transmisión y adquisición de imagen o video, también se caracteriza por ser pequeño y sobre todo bastante económico, siendo su precio no mayor a 7 dólares.

A continuación, algunas características generales del microcontrolador y la descripción general de los puertos del microcontrolador:

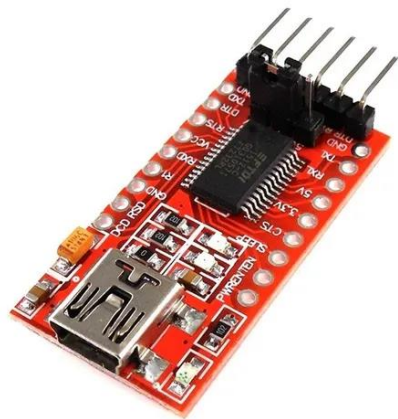
Características del ESP32-CAM	
<b>Procesador</b>	Procesador de 32 bits de 160 MHz
<b>RAM</b>	520 KB SRAM integrada, expandible 4 MPSRAM
<b>Puertos soportados</b>	UART/SPI/I2C/PWM/ADC/DAC
<b>Tecnologías soportadas</b>	TFCards, OV2640 and OV7670 cameras



*Imagen 2. Características generales del ESP-32*

Puerto	Tipo	Descripción
<b>GPIO 4</b>	I/O	Datos 1 aunque también se puede emplear para controlar el LED que está integrado en el ESP32
<b>GPIO 2, 12 y 13</b>	I/O	Puertos de entrada y salida de datos
<b>GPIO 1 y 3</b>	TXD y RXD	Son los pines del puerto serial por lo que se usan para cargar el código en memoria
<b>GPIO 0</b>	MCLK	Además de ser de entrada y salida de datos sirve también para determinar los modos del microcontrolador
<b>GPIO 14 y 15</b>	CLK y CMD	Además de ser puertos de entrada y salida de datos, también pueden ser usados para realizar acciones a más bajo nivel

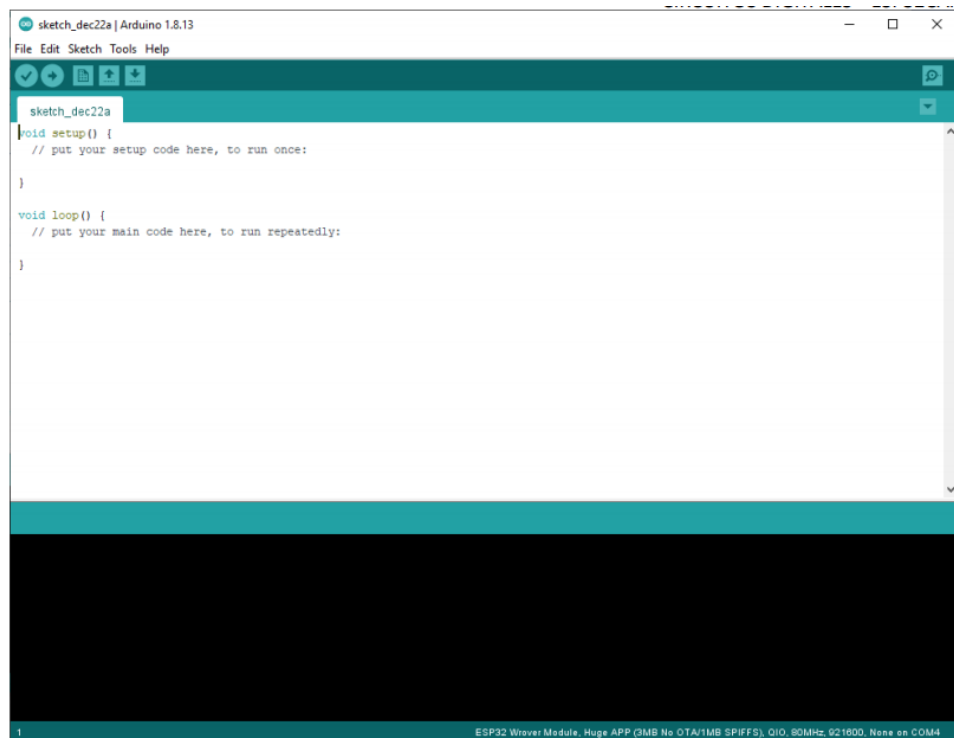
Por último, para poder descargar código a este microcontrolador es necesario emplear un programador-usb UART (FTDI),:



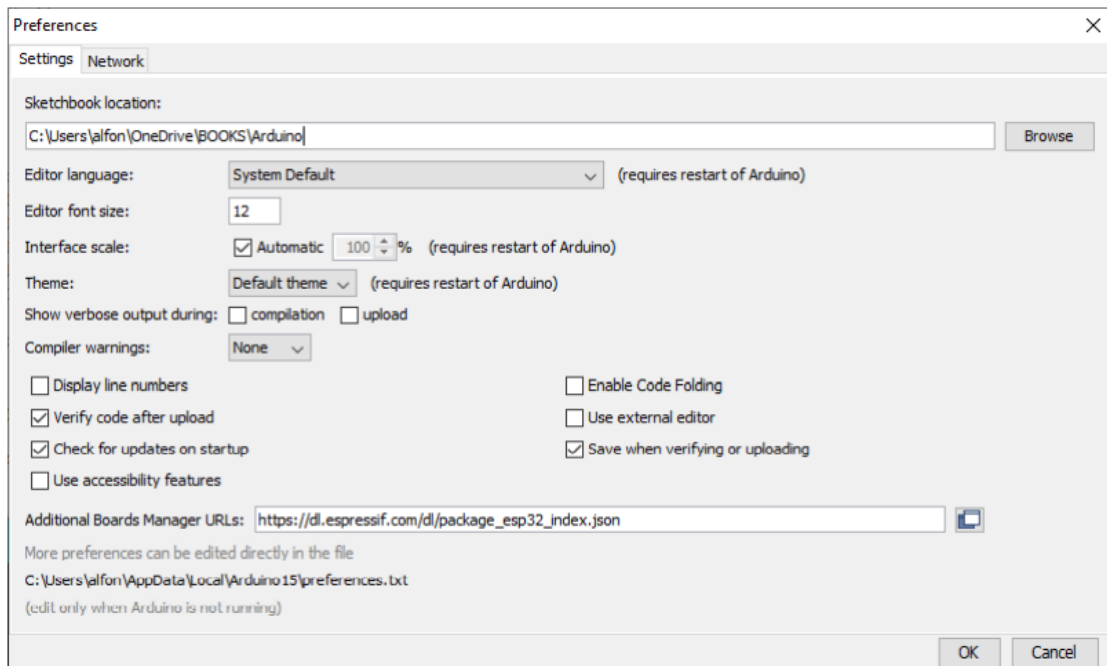
*Imagen 3. Programador UART*

### ***3. Conceptos de IDE y lanzador de código***

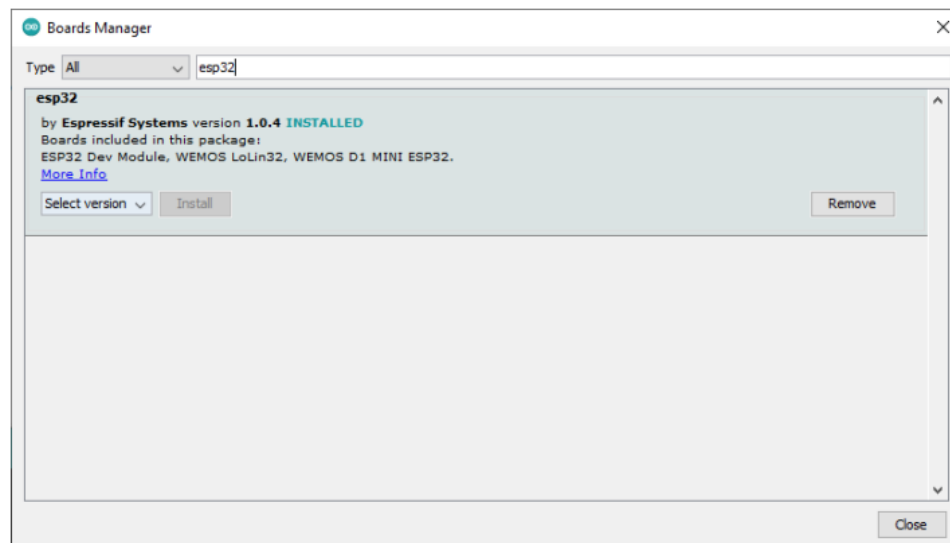
Una vez conocido el hardware, es necesario configurar el ambiente de desarrollo que emplearemos para programar, compilar y descargar nuestros programas. En este caso para poder compilar y descargar el código a nuestro microcontrolador se empleará Arduino IDE, en caso de no tenerlo descárgalo de la página oficial de Arduino; <https://www.arduino.com>



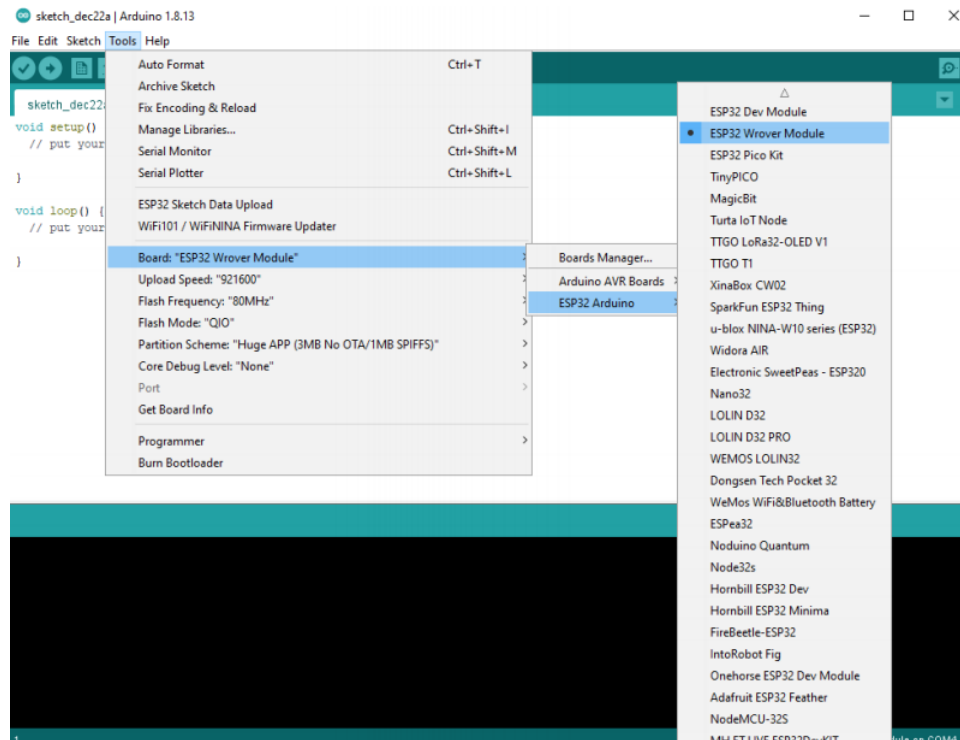
Ya instalado el IDE, lo primero que debemos hacer es descargar las bibliotecas de ESP32 necesarias para trabajar, para ello en el apartado de Arduino > Preferences, agregar el siguiente URL [https://dl.espressif.com/dl/package\\_esp32\\_index.json](https://dl.espressif.com/dl/package_esp32_index.json) en el apartado de “Additional Board Manager URLs”



Posteriormente, ir al apartado de **Tools > Board > Boards > Manager**, escribir o buscar ESP32 e instalar la biblioteca resultante:



Una vez instalada escoger la tarjeta a la que vamos a descargar nuestro programa, específicamente es la denominada “**ESP32 Wrover Module**”



## Desarrollo

### 1. Arquitectura y Diagrama de Componentes

Como previamente se ha mencionado en el presente proyecto se emplearán diversas tecnologías que están conectadas entre sí para poder llevar a cabo un detector de objetos, a continuación, se muestra el diagrama de componentes:

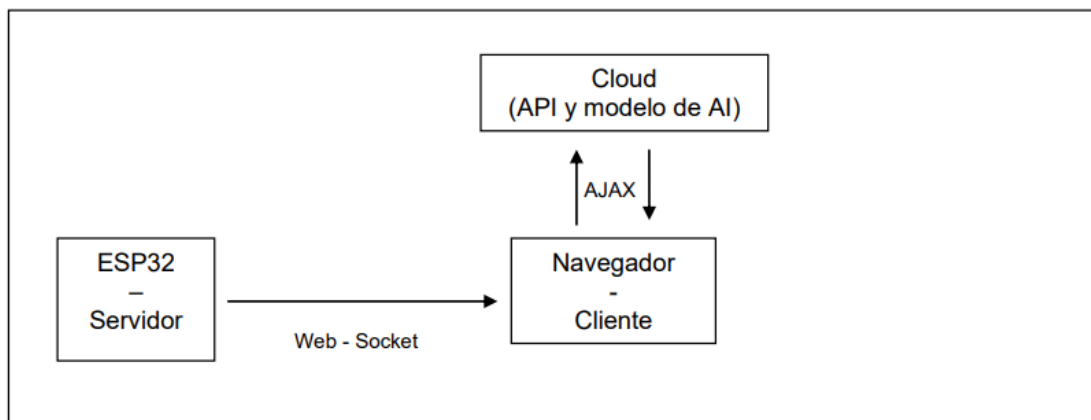


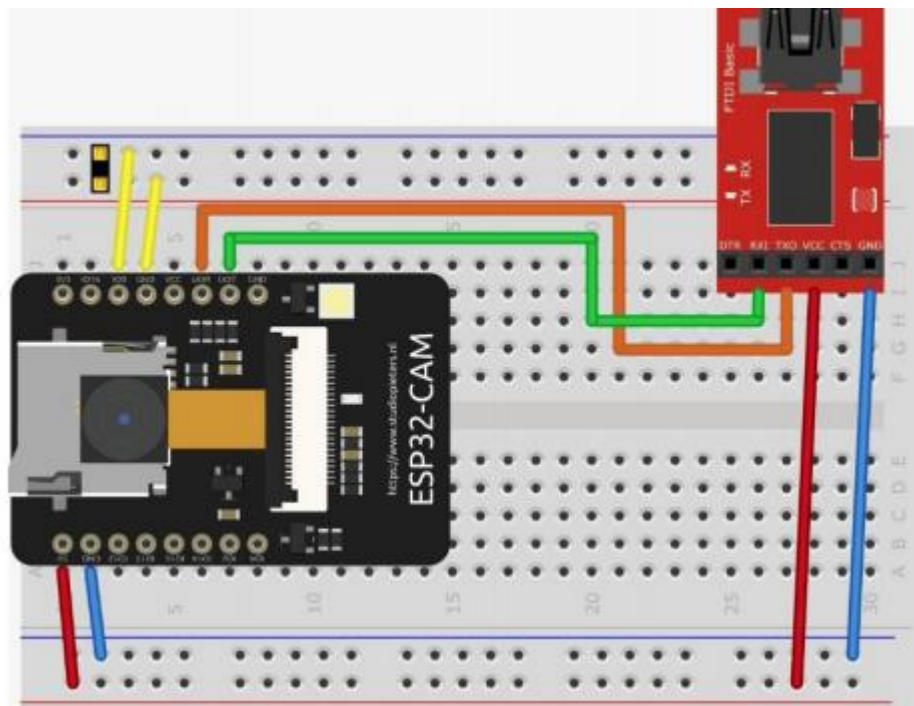
Imagen 4. Diagrama de componentes del sistema de detección de objetos



Con base a la imagen 1, podemos observar una conexión entre nuestro servidor con el cliente mediante un web socket esto con el objetivo de transmitir el video adquirido mediante la cámara integrada en nuestro microcontrolador, además de que en el mismo servidor deberá declararse las llamadas a la API a través del uso de AJAX.

## ***2. Diagrama de conexiones***

Entre las muchas ventajas que nos ofrece el ESP32-CAM es un módulo de cámara\*, específicamente el modelo cam-OV2640, sin embargo, debido a lo compacto que es podemos observar que no integra una conexión directa a USB, por lo que se usará un programador FTDI para de esta forma descargar nuestros proyectos en memoria. A continuación, se muestra el esquema de cómo se conectaría el ESP32 con el programador FTDI:



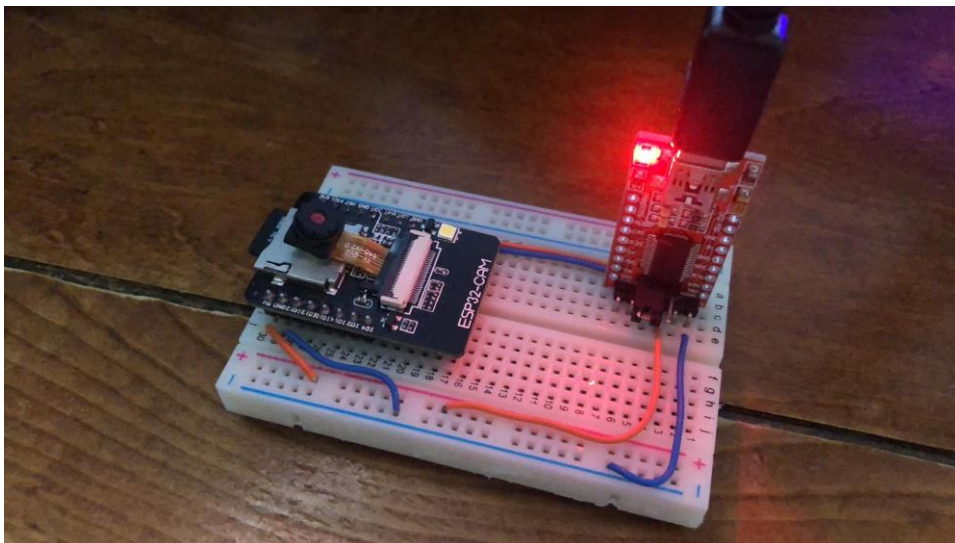
*Imagen 5. Diagrama de conexiones entre componentes de hardware*

Podemos observar que tenemos usados los puertos correspondientes GND y alimentación de 5V (Azul y rojo), a su vez observamos que se encuentran conectados el

GPIO1 y GPIO3 (Naranja y verde) que serán los puertos para descargar nuestros futuros programas, por otro lado, debido a que vamos a grabar nuestro programa en memoria necesitamos conectar GPIO 0 a GND en este caso se empleó directamente el que está al lado del pin 0 (Amarillo).

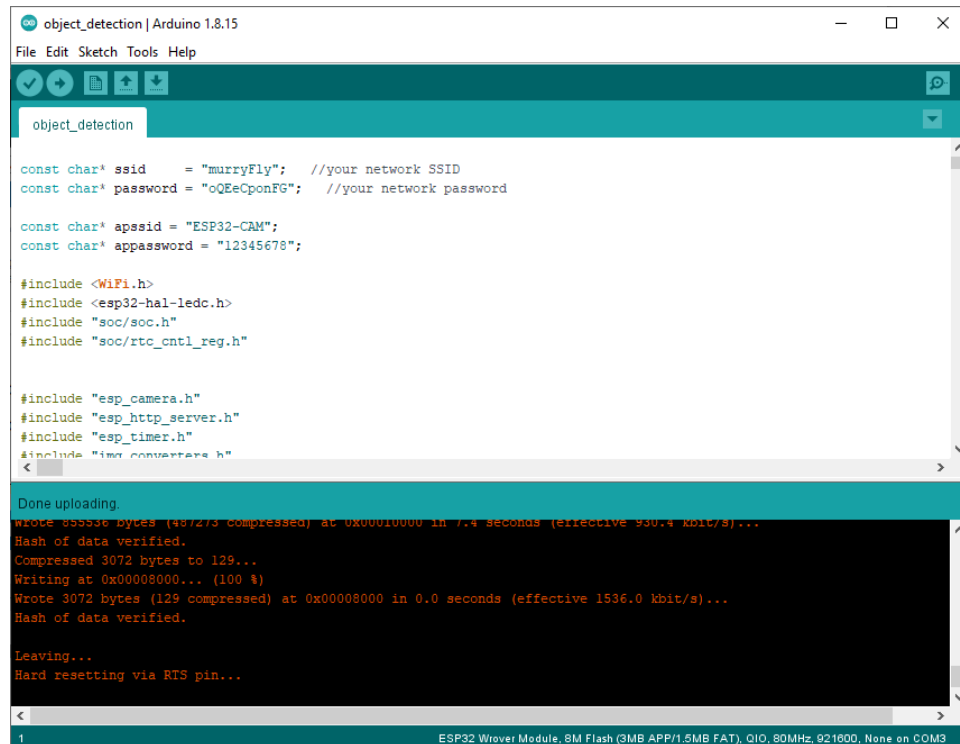
### ***3. Compilación y descarga del programa***

Para compilar y descargar el programa a nuestro microcontrolador lo primero que debemos hacer es verificar las conexiones a nuestro micro y ponerlo en corto mediante sus puertos GND y GPIO0 (Ver imagen 5). A continuación, se apreciar una fotografía de los componentes ya conectados en físico:



*Imagen 6. Componentes en físico del proyecto*

Posteriormente compilamos y lanzamos en nuestro IDE de preferencia en nuestro caso el IDE de Arduino con las bibliotecas y configuraciones previamente descritas:



```
object_detection | Arduino 1.8.15
File Edit Sketch Tools Help

object_detection

const char* ssid = "murryFly"; //your network SSID
const char* password = "oQEeCponFG"; //your network password

const char* apssid = "ESP32-CAM";
const char* appassword = "12345678";

#include <WiFi.h>
#include <esp32-hal-ledc.h>
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"

#include "esp_camera.h"
#include "esp_http_server.h"
#include "esp_timer.h"
#include "img_converters.h"

Done uploading.
wrote 65536 bytes (4672/3 compressed) at 0x00000000 in 7.4 seconds (effective 930.4 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 129...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (129 compressed) at 0x00008000 in 0.0 seconds (effective 1536.0 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...

ESP32 Wrover Module, 8M Flash (3MB APP/1.5MB FAT), QIO, 80MHz, 921800, None on COM3
```

Una vez descargado el programa, antes de oprimir el botón de reset es necesario desconectar el GPIO0 del GND (cables amarillos), en caso de no hacerlo en el monitor serial de nuestro IDE se mostraría un mensaje de “programa por descargar”.

#### 4. Descripción del programa

Como previamente hemos visto en el diagrama de componentes (Imagen 4), nuestro programa a montar en el ESP32 contiene 2 principales segmentos o entidades:

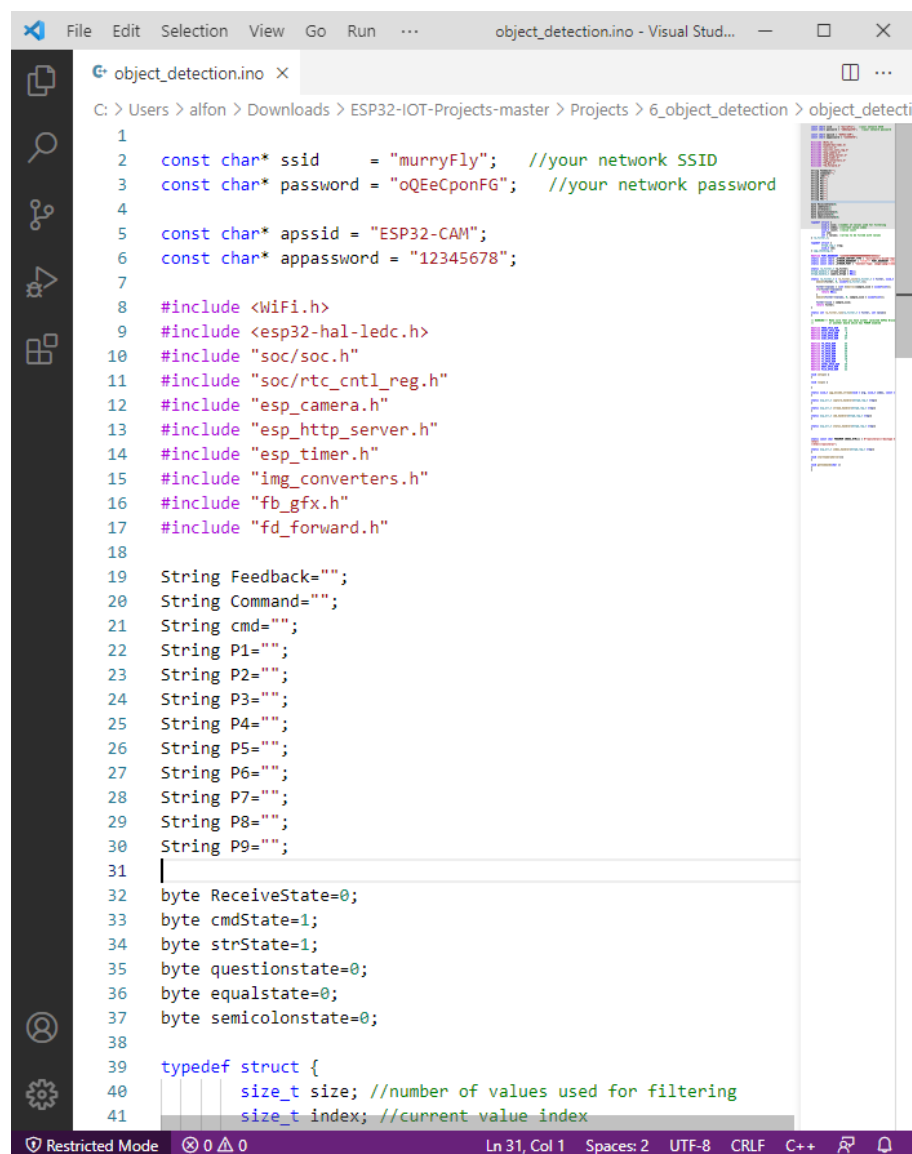
1. Un web-scket para la conexión mediante una API con TensorFlow
2. Un segmento de configuraciones de la cámara para la decodificación del stream de vídeo y *stream* de video

A continuación, se explican parte a parte los segmentos de código del programa-proyecto:

## Bibliotecas y variables globales

La primera parte que podemos observar es el uso de constantes para el manejo de los ID y las contraseñas tanto del internet local como del programa mismo. Por otro lado, observamos la importación de todas las bibliotecas empleadas en el mismo programa:

Para esta parte es necesario mencionar que al estar basado en el programa raíz de web cam por parte de expressif, por ello observamos el uso de bibliotecas como “esp\_camera.h” o “wifi.h”

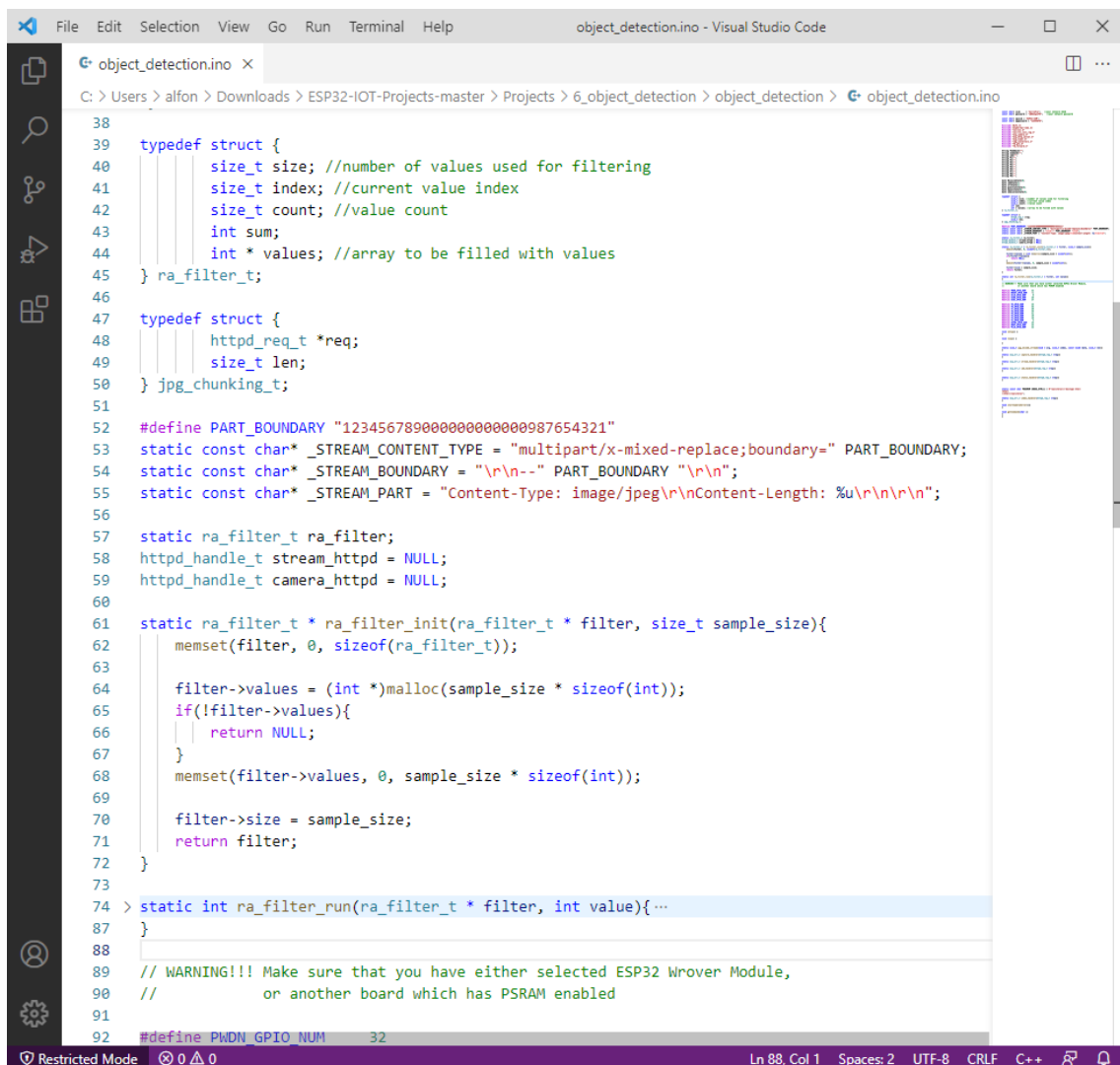


```
1
2  const char* ssid      = "murryFly"; //your network SSID
3  const char* password  = "oQEeCponFG"; //your network password
4
5  const char* apssid = "ESP32-CAM";
6  const char* appassword = "12345678";
7
8  #include <WiFi.h>
9  #include <esp32-hal-ledc.h>
10 #include "soc/soc.h"
11 #include "soc/rtc_cntl_reg.h"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

Por último, cabe mencionar que todas las variables de tipo byte son meramente para contemplar los diferentes estados que a lo largo deben activarse para poder conectar el stream de video captado por la cámara con una página que además estará integrada con los resultados pos-procesamiento del modelo de Inteligencia Artificial.

## Definición de estructuras (Objetos) y funciones estáticas para manejo y filtrado de datos del sensor de la cámara

Para este apartado observamos 2 principales estructuras de datos u objetos que están destinados al manejo del filtro aplicado al otro objeto que es el chunking de imagen al adquirir datos el sensor de la cámara.



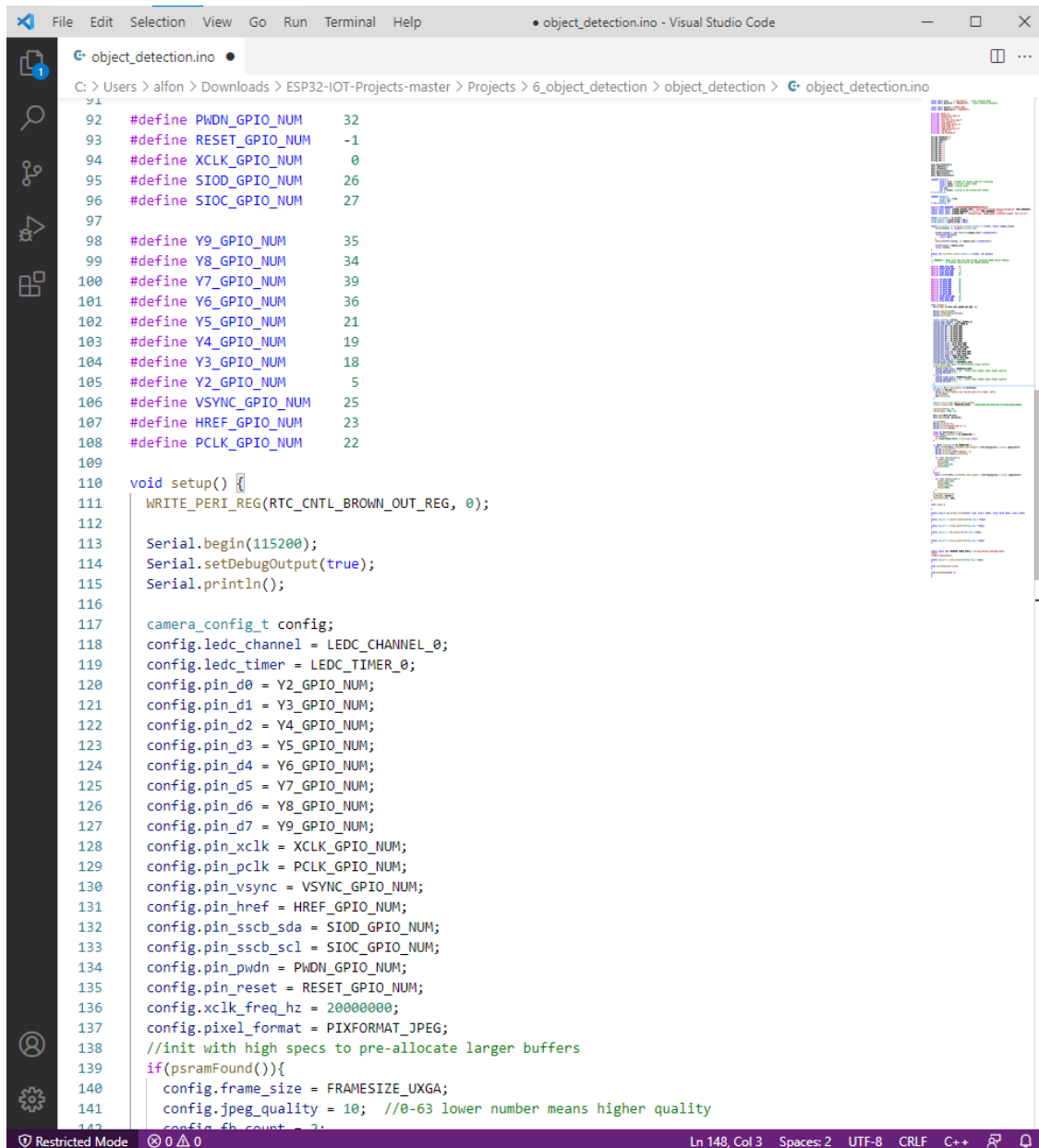
```
38
39     typedef struct {
40         size_t size; //number of values used for filtering
41         size_t index; //current value index
42         size_t count; //value count
43         int sum;
44         int * values; //array to be filled with values
45     } ra_filter_t;
46
47     typedef struct {
48         httpd_req_t *req;
49         size_t len;
50     } jpg_chunking_t;
51
52     #define PART_BOUNDARY "1234567890000000000000987654321"
53     static const char* _STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;boundary=" PART_BOUNDARY;
54     static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
55     static const char* _STREAM_PART = "Content-Type: image/jpeg\r\nContent-Length: %u\r\n\r\n";
56
57     static ra_filter_t ra_filter;
58     httpd_handle_t stream_httpd = NULL;
59     httpd_handle_t camera_httpd = NULL;
60
61     static ra_filter_t * ra_filter_init(ra_filter_t * filter, size_t sample_size){
62         memset(filter, 0, sizeof(ra_filter_t));
63
64         filter->values = (int *)malloc(sample_size * sizeof(int));
65         if(!filter->values){
66             return NULL;
67         }
68         memset(filter->values, 0, sample_size * sizeof(int));
69
70         filter->size = sample_size;
71         return filter;
72     }
73
74     static int ra_filter_run(ra_filter_t * filter, int value){...
75 }
76
77 // WARNING!!! Make sure that you have either selected ESP32 Wrover Module,
78 // or another board which has PSRAM enabled
79
80 #define PWDN_GPIO_NUM 32
```

Por otro lado, observamos las 2 primeras funciones desarrolladas por expressif para el manejo del chunking de imagen y el filtro de la imagen cruda adquirida por el sensor de la cámara.

### **Definición de puertos y configuración - asignación de variables**

Como bien sabemos, en la gran mayoría de proyectos con microcontroladores, tenemos de manera general 2 funciones void generales, el void setup dedicado a la configuración del microcontrolador como la función void loop para todas las acciones a realizar por nuestro micro.

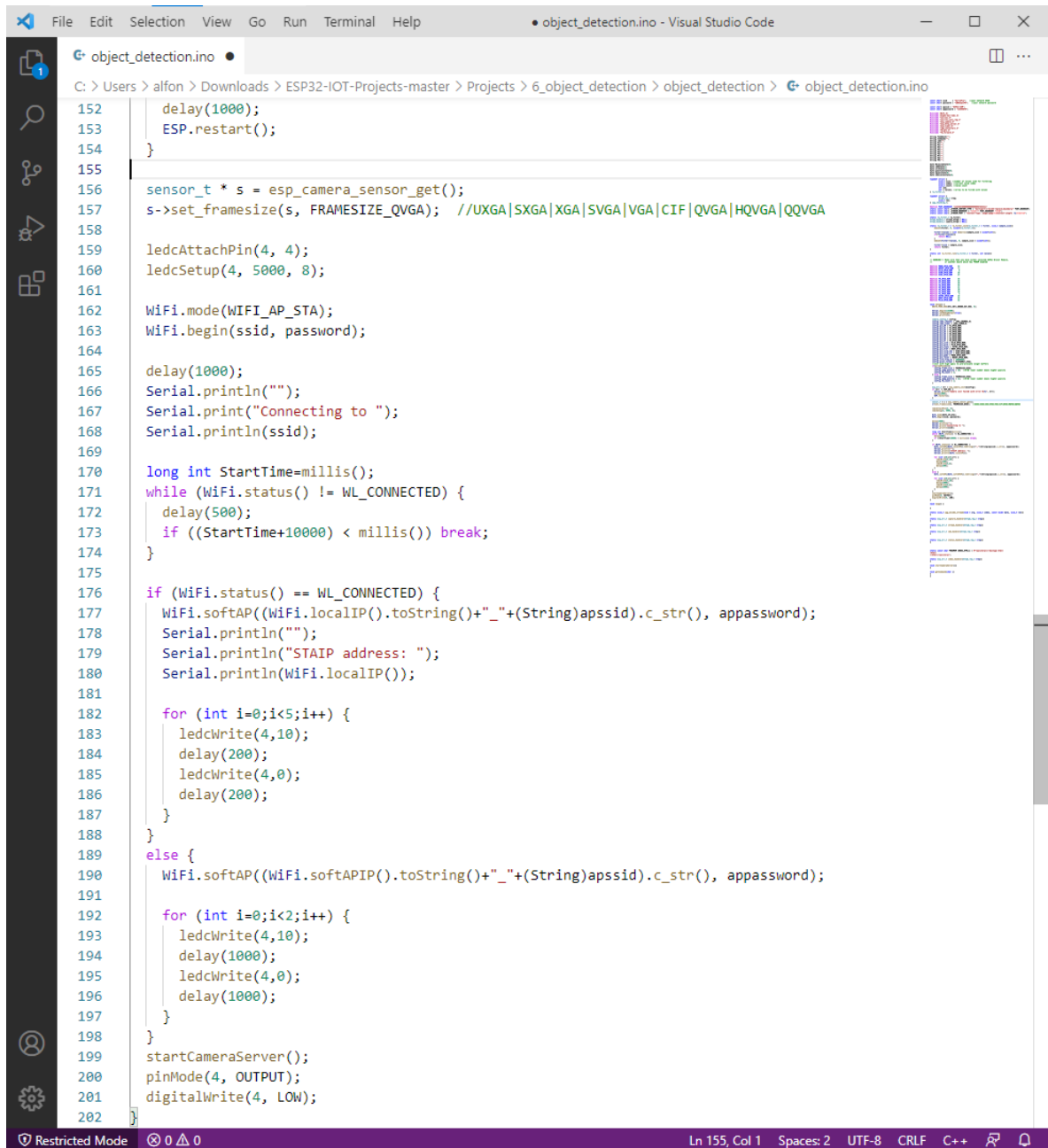
En el código inferior observamos la declaración de los puertos para el manejo de la cámara y a su vez algunos valores designados para el manejo de la misma.



```
object_detection.ino
C:\Users> alfon > Downloads > ESP32-IOT-Projects-master > Projects > 6_object_detection > object_detection > object_detection.ino
92 #define PWDN_GPIO_NUM 32
93 #define RESET_GPIO_NUM -1
94 #define XCLK_GPIO_NUM 0
95 #define SIOD_GPIO_NUM 26
96 #define SIOC_GPIO_NUM 27
97
98 #define Y9_GPIO_NUM 35
99 #define Y8_GPIO_NUM 34
100 #define Y7_GPIO_NUM 39
101 #define Y6_GPIO_NUM 36
102 #define Y5_GPIO_NUM 21
103 #define Y4_GPIO_NUM 19
104 #define Y3_GPIO_NUM 18
105 #define Y2_GPIO_NUM 5
106 #define VSYNC_GPIO_NUM 25
107 #define HREF_GPIO_NUM 23
108 #define PCLK_GPIO_NUM 22
109
110 void setup() {
111     WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
112
113     Serial.begin(115200);
114     Serial.setDebugOutput(true);
115     Serial.println();
116
117     camera_config_t config;
118     config.ledc_channel = LEDC_CHANNEL_0;
119     config.ledc_timer = LEDC_TIMER_0;
120     config.pin_d0 = Y2_GPIO_NUM;
121     config.pin_d1 = Y3_GPIO_NUM;
122     config.pin_d2 = Y4_GPIO_NUM;
123     config.pin_d3 = Y5_GPIO_NUM;
124     config.pin_d4 = Y6_GPIO_NUM;
125     config.pin_d5 = Y7_GPIO_NUM;
126     config.pin_d6 = Y8_GPIO_NUM;
127     config.pin_d7 = Y9_GPIO_NUM;
128     config.pin_xclk = XCLK_GPIO_NUM;
129     config.pin_pclk = PCLK_GPIO_NUM;
130     config.pin_vsync = VSYNC_GPIO_NUM;
131     config.pin_href = HREF_GPIO_NUM;
132     config.pin_sscb_sda = SIOD_GPIO_NUM;
133     config.pin_sscb_scl = SIOC_GPIO_NUM;
134     config.pin_pwdn = PWDN_GPIO_NUM;
135     config.pin_reset = RESET_GPIO_NUM;
136     config.xclk_freq_hz = 20000000;
137     config.pixel_format = PIXFORMAT_JPEG;
138     //init with high specs to pre-allocate larger buffers
139     if(psramFound()){
140         config.frame_size = FRAMESIZE_UXGA;
141         config.jpeg_quality = 10; //0-63 lower number means higher quality
142         config.fb_count = 2;
```

## Configuración del WiFi y llamadas a funciones estáticas

Para la configuración de nuestro web socket previamente necesitamos hacer uso de nuestro wifi interno, a continuación, se muestra la instancia del objeto traído de la biblioteca wifi.h además de las configuraciones a este mismo



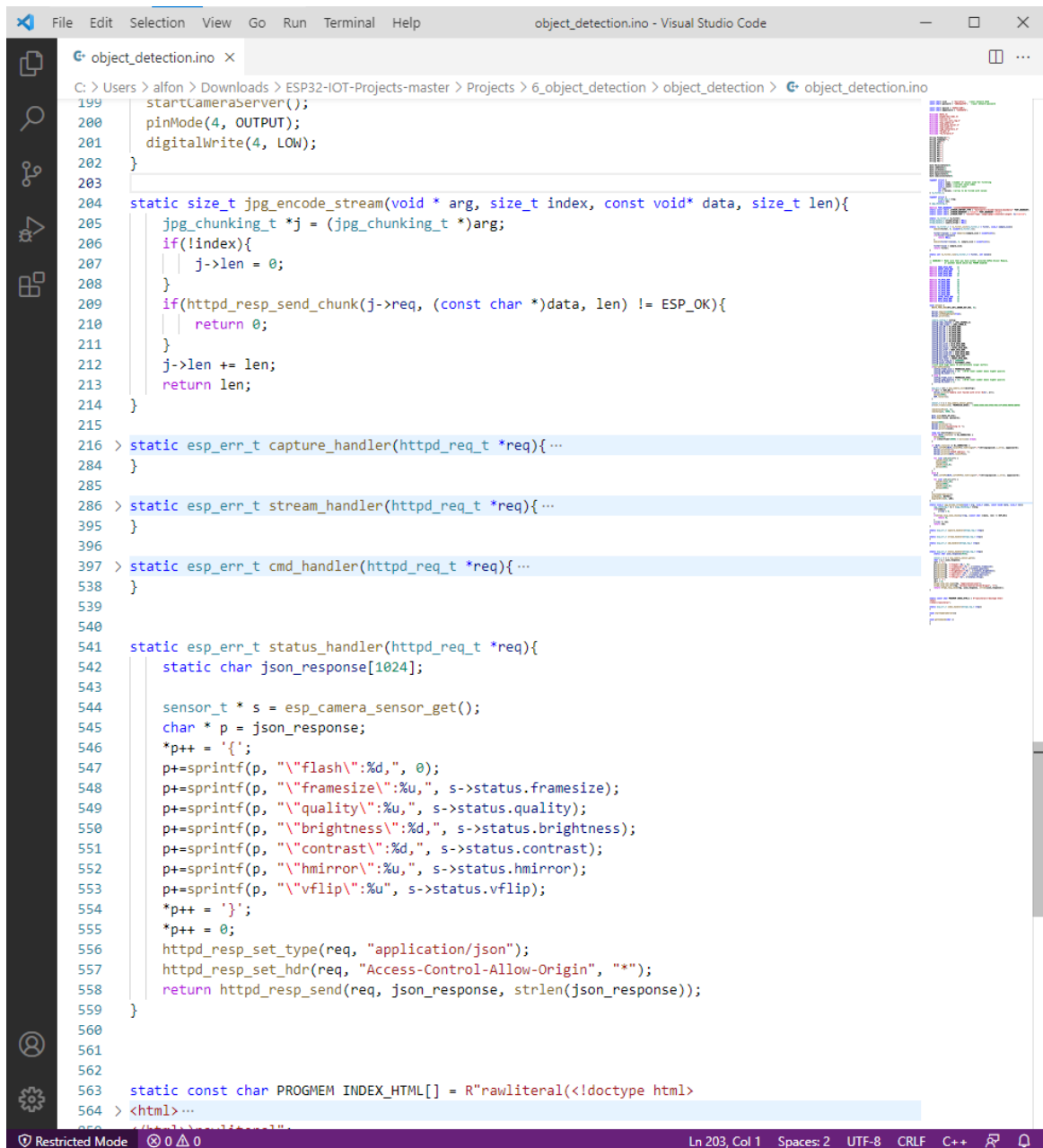
```
152     delay(1000);
153     ESP.restart();
154 }
155
156 sensor_t * s = esp_camera_sensor_get();
157 s->set_framesize(s, FRAMESIZE_QVGA); //UXGA|SXGA|XGA|SVGA|VGA|CIF|QVGA|HQVGA|QQVGA
158
159 ledcAttachPin(4, 4);
160 ledcSetup(4, 5000, 8);
161
162 WiFi.mode(WIFI_AP_STA);
163 WiFi.begin(ssid, password);
164
165 delay(1000);
166 Serial.println("");
167 Serial.print("Connecting to ");
168 Serial.println(ssid);
169
170 long int StartTime=millis();
171 while (WiFi.status() != WL_CONNECTED) {
172     delay(500);
173     if ((StartTime+10000) < millis()) break;
174 }
175
176 if (WiFi.status() == WL_CONNECTED) {
177     WiFi.softAP((WiFi.localIP().toString()+"_"+(String)apssid).c_str(), appassword);
178     Serial.println("");
179     Serial.println("STAIP address: ");
180     Serial.println(WiFi.localIP());
181
182     for (int i=0;i<5;i++) {
183         ledcWrite(4,10);
184         delay(200);
185         ledcWrite(4,0);
186         delay(200);
187     }
188 }
189 else {
190     WiFi.softAP((WiFi.softAPIP().toString()+"_"+(String)apssid).c_str(), appassword);
191
192     for (int i=0;i<2;i++) {
193         ledcWrite(4,10);
194         delay(1000);
195         ledcWrite(4,0);
196         delay(1000);
197     }
198 }
199 startCameraServer();
200 pinMode(4, OUTPUT);
201 digitalWrite(4, LOW);
202 }
```

## Funciones estáticas para el manejo y configuración de la cámara

Como bien se ha mencionado previamente, expressif ya nos proporciona bloques de código previos con el propósito de poder usar directamente el stream de video de nuestro microcontrolador. Las siguientes 4 funciones son las encargadas precisamente de todo el proceso digital por el que debe pasar una imagen desde que es capturada y transformada a



su versión cruda o RAW hasta la conversión, filtrado y compresión en un formato como jpeg.



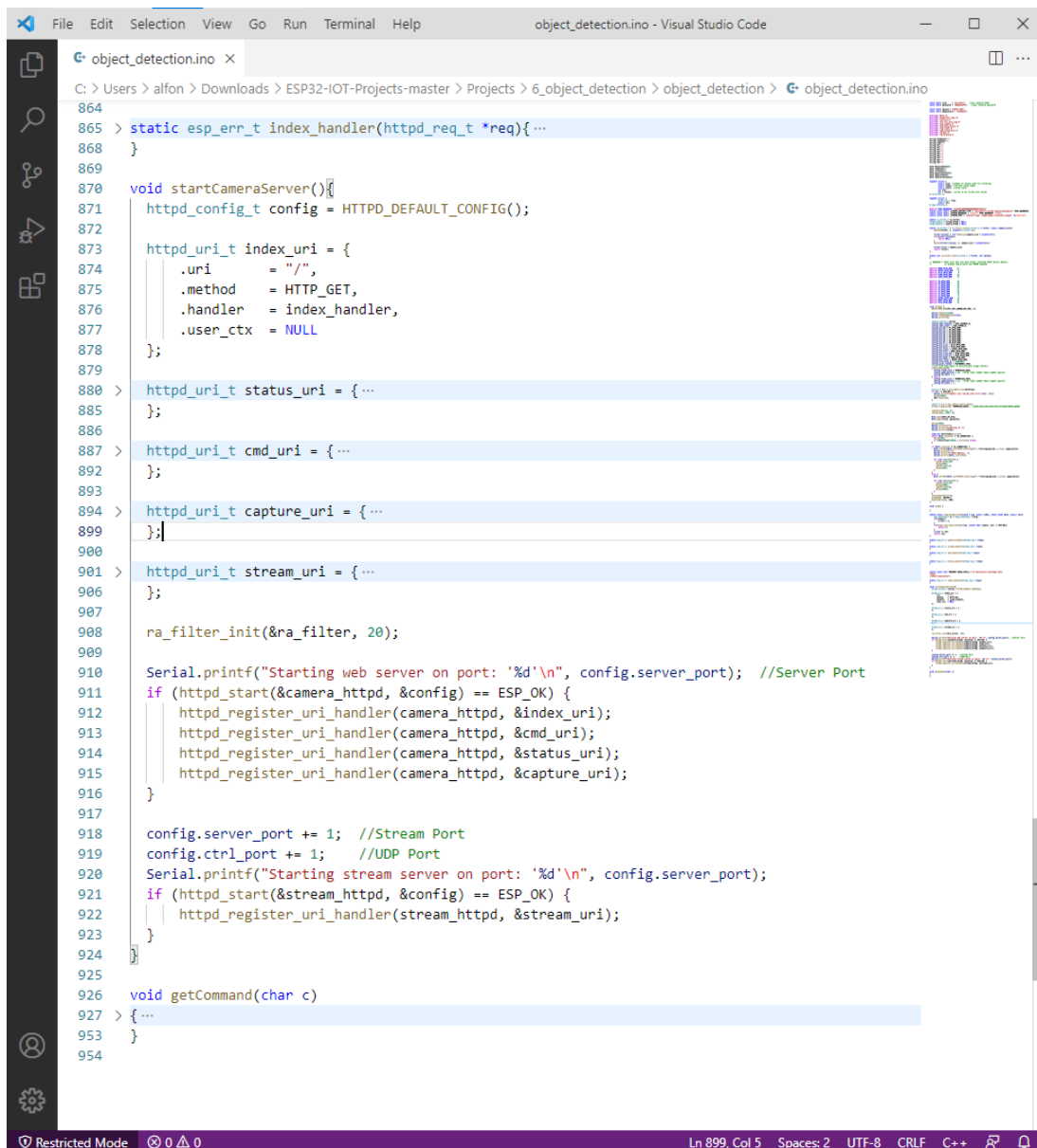
```
199 startCameraServer();
200 pinMode(4, OUTPUT);
201 digitalWrite(4, LOW);
202 }
203
204 static size_t jpg_encode_stream(void * arg, size_t index, const void* data, size_t len){
205     jpg_chunking_t *j = (jpg_chunking_t *)arg;
206     if(!index){
207         j->len = 0;
208     }
209     if(httpd_resp_send_chunk(j->req, (const char *)data, len) != ESP_OK){
210         return 0;
211     }
212     j->len += len;
213     return len;
214 }
215
216 > static esp_err_t capture_handler(httpd_req_t *req){ ...
284 }
285
286 > static esp_err_t stream_handler(httpd_req_t *req){ ...
395 }
396
397 > static esp_err_t cmd_handler(httpd_req_t *req){ ...
538 }
539
540
541 static esp_err_t status_handler(httpd_req_t *req){
542     static char json_response[1024];
543
544     sensor_t * s = esp_camera_sensor_get();
545     char * p = json_response;
546     *p++ = '{';
547     p+=sprintf(p, "\"flash\":%d,", s->status.flash, 0);
548     p+=sprintf(p, "\"framesize\":%u,", s->status.framesize);
549     p+=sprintf(p, "\"quality\":%u,", s->status.quality);
550     p+=sprintf(p, "\"brightness\":%d,", s->status.brightness);
551     p+=sprintf(p, "\"contrast\":%d,", s->status.contrast);
552     p+=sprintf(p, "\"hmirror\":%u,", s->status.hmirror);
553     p+=sprintf(p, "\"vflip\":%u", s->status.vflip);
554     *p++ = '}';
555     *p++ = 0;
556     httpd_resp_set_type(req, "application/json");
557     httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
558     return httpd_resp_send(req, json_response, strlen(json_response));
559 }
560
561
562
563 static const char PROGMEM INDEX_HTML[] = R"rawliteral(<!doctype html>
564 > <html> ...
565 rawliteral(")";
```

## Funciones para el manejo del protocolo HTTP y web socket

Como bien sabemos un web socket es un componente creado exclusivamente como puente o medio de comunicación entre lo que tenemos como datos adquiridos por nuestro microcontrolador y lo que será nuestra futura conexión a internet o específicamente

consumir o mandar datos a nuestro cliente conectado al servidor (En este caso el mismo microcontrolador).

Podemos ver que la función *startCameraServer* es la encargada precisamente de instanciar el uri empleado para poder acceder mediante http a nuestro sitio web, esto evidentemente empleando una LAN además usar como puente de comunicación el web socket declarado en la función *index\_handler*.



```
864 > static esp_err_t index_handler(httpd_req_t *req){...
868 }
869
870 void startCameraServer(){
871     httpd_config_t config = HTTPD_DEFAULT_CONFIG();
872
873     httpd_uri_t index_uri = {
874         .uri       = "/",
875         .method    = HTTP_GET,
876         .handler    = index_handler,
877         .user_ctx   = NULL
878     };
879
880 > httpd_uri_t status_uri = { ...
885 };
886
887 > httpd_uri_t cmd_uri = { ...
892 };
893
894 > httpd_uri_t capture_uri = { ...
899 };
900
901 > httpd_uri_t stream_uri = { ...
906 };
907
908 ra_filter_init(&ra_filter, 20);
909
910 Serial.printf("Starting web server on port: '%d'\n", config.server_port); //Server Port
911 if (httpd_start(&camera_httpd, &config) == ESP_OK) {
912     httpd_register_uri_handler(camera_httpd, &index_uri);
913     httpd_register_uri_handler(camera_httpd, &cmd_uri);
914     httpd_register_uri_handler(camera_httpd, &status_uri);
915     httpd_register_uri_handler(camera_httpd, &capture_uri);
916 }
917
918 config.server_port += 1; //Stream Port
919 config.ctrl_port += 1; //UDP Port
920 Serial.printf("Starting stream server on port: '%d'\n", config.server_port);
921 if (httpd_start(&stream_httpd, &config) == ESP_OK) {
922     httpd_register_uri_handler(stream_httpd, &stream_uri);
923 }
924
925
926 void getCommand(char c)
927 > { ...
953 }
954
```

## Página web y conexión al API – Modelo de IA

Como bien sabemos, el código generalmente empleado para programar microcontroladores suele estar escrito en C, C++ o ensamblador si bien algunos casos también pueden estar escritos en Micropython, al final en el mismo IDE se hace un *parsing* para pasar a un lenguaje como C.

Es por ello que para poder hacer una página web y la debida conexión a nuestro API y modelo de inteligencia artificial debemos guardar todo el código HTML en una variable interpretable por C y específicamente por el objeto de web socket.

A continuación, la forma en que fue guardado todo el código dedicado a este apartado:

```
567 > static const char PROGMEM INDEX_HTML[] = R"rawliteral(<!doctype html> ...  
863 </html>)rawliteral";  
864
```

Sin embargo, y como se puede apreciar en la diferencia de líneas, realmente este segmento de código lleva a cabo 3 aspectos en concreto:

1. La creación de una página web sencilla con el objetivo de poder desplegar toda la información procesada por el modelo de AI.

En la imagen inferior se puede apreciar en la etiqueta **body** , una parte del segmento del menú encargado de configurar la forma en que es presentada la información ya procesada por el modelo de IA:



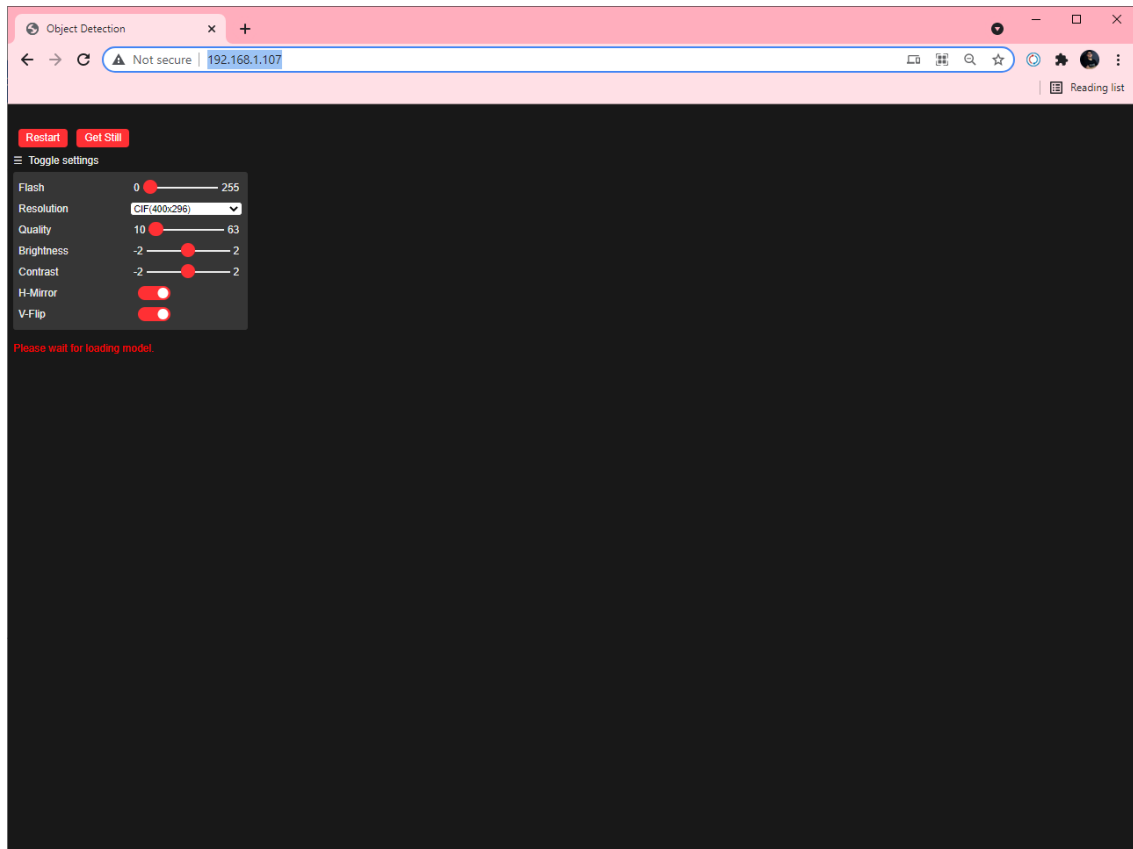
```
218
219
220 <script>
221 var restart = document.getElementById('restart');
222 var getStill = document.getElementById('get-still');
223 var ShowImage = document.getElementById('stream');
224 var canvas = document.getElementById("canvas");
225 var context = canvas.getContext("2d");
226 var result = document.getElementById('result');
227 var Model;
228
229 function LoadModel() {
230     result.innerHTML = "Please wait for loading model.";
231     cocoSsd.load().then(cocoSsd_Model => {
232         Model = cocoSsd_Model;
233         result.innerHTML = "";
234         getStill.style.display = "block";
235         getStill.click();
236     });
237 }
238
239 function DetectImage() {
240     canvas.setAttribute("width", ShowImage.width);
241     canvas.setAttribute("height", ShowImage.height);
242     context.drawImage(ShowImage, 0, 0, ShowImage.width, ShowImage.height);
243
244     Model.detect(canvas).then(Predictions => {
245         var s = (ShowImage.width>ShowImage.height)?ShowImage.width:ShowImage.height;
246
247         //console.log('Predictions: ', Predictions);
248         if (Predictions.length>0) {
249             result.innerHTML = "";
250             for (var i=0;i<Predictions.length;i++) {
251                 const x = Predictions[i].bbox[0];
252                 const y = Predictions[i].bbox[1];
253                 const width = Predictions[i].bbox[2];
254                 const height = Predictions[i].bbox[3];
255                 context.lineWidth = Math.round(s/200);
256                 context.strokeStyle = "#00FFFF";
257                 context.beginPath();
258                 context.rect(x, y, width, height);
259                 context.stroke();
260                 context.lineWidth = "2";
261                 context.fillStyle = "red";
262                 context.font = Math.round(s/30) + "px Arial";
263                 context.fillText(Predictions[i].class, x, y);
264                 //context.fillText(i, x, y);
265                 result.innerHTML+= "[ "+i+" ] "+Predictions[i].class+", "+Math.round(Predictions[i].score*100)+"%, "+Math.round(
266             }
267
268             for (var j=0;j<Predictions.length;j++) {
269                 //https://github.com/tensorflow/tfjs-models/blob/master/coco-ssd/src/classes.ts
270                 if (Predictions[j].class=="person"&&Predictions[j].score>0.5) {
271                     try{
```

3. La llamada a la API para poder obtener los datos ya analizados por la AI, para de esa forma mostrarla al cliente, a continuación, se muestra precisamente un pequeño código en JavaScript para hacer usar específico de este modelo, donde se muestra el cómo es que se hará uso de la información que se mande con su posterior muestra en el sitio web:

```
95 <div id="result" style="color:red">Please wait for loading model.</div>
96
97 <script>
98   document.addEventListener('DOMContentLoaded', function (event) {
99     var baseHost = document.location.origin
100     var streamUrl = baseHost + ':81'
101     const hide = el => {...}
102     const show = el => {...}
103     const disable = el => {...}
104     const enable = el => {...}
105     const updateValue = (el, value, updateRemote) => {
106       updateRemote = updateRemote == null ? true : updateRemote
107       let initialValue
108       if (el.type === 'checkbox') {
109         initialValue = el.checked
110         value = !value
111         el.checked = value
112       } else {
113         initialValue = el.value
114         el.value = value
115       }
116       if (updateRemote && initialValue !== value) {
117         updateConfig(el);
118       }
119     }
120     function updateConfig (el) {
121       let value
122       switch (el.type) {
123         case 'checkbox':
124           value = el.checked ? 1 : 0
125           break
126         case 'range':
127         case 'select-one':
128           value = el.value
129           break
130         case 'button':
131         case 'submit':
132           value = '1'
133           break
134         default:
135           return
136       }
137       const query = `${baseHost}/control?var=${el.id}&val=${value}`
138       fetch(query)
139         .then(response => {
140           console.log('request to ${query} finished, status: ${response.status}')
141         })
142     }
143     document
144       .querySelectorAll('.close')
145       .forEach(el => {
146         el.onclick = () => {
147
```

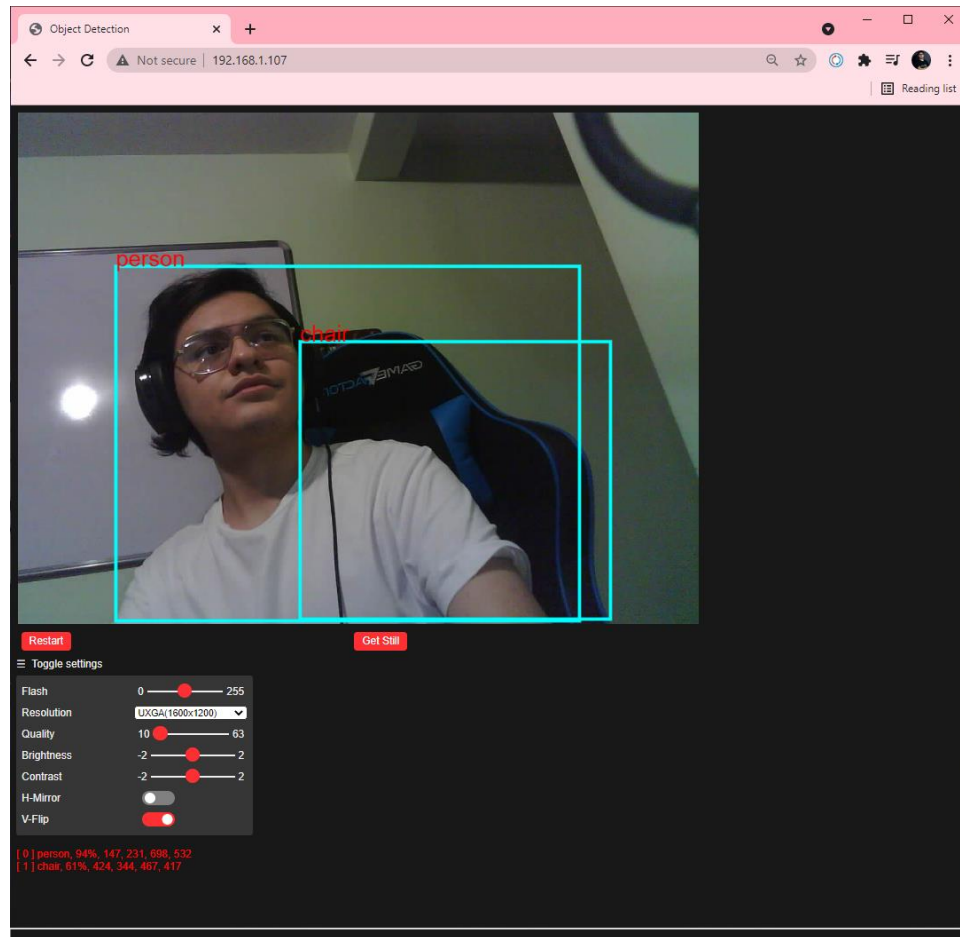
## Resultados finales

Con base a la descripción detallada de cómo es que fue llamado e implementado un modelo de Inteligencia Artificial pre-entrenado, es como a continuación se muestran los resultados obtenidos:



*Imagen 7. Muestra de la página web e instancia del web socket*

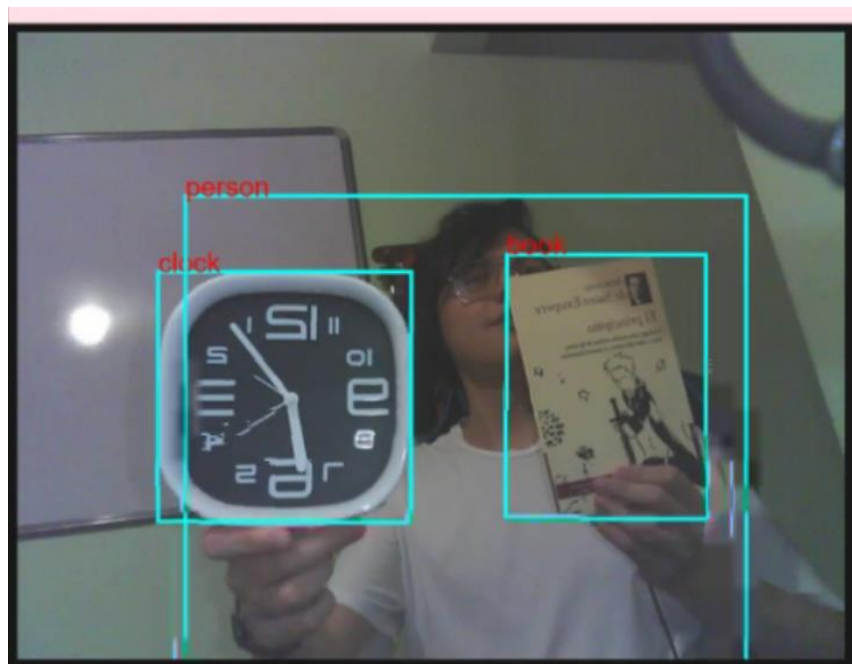
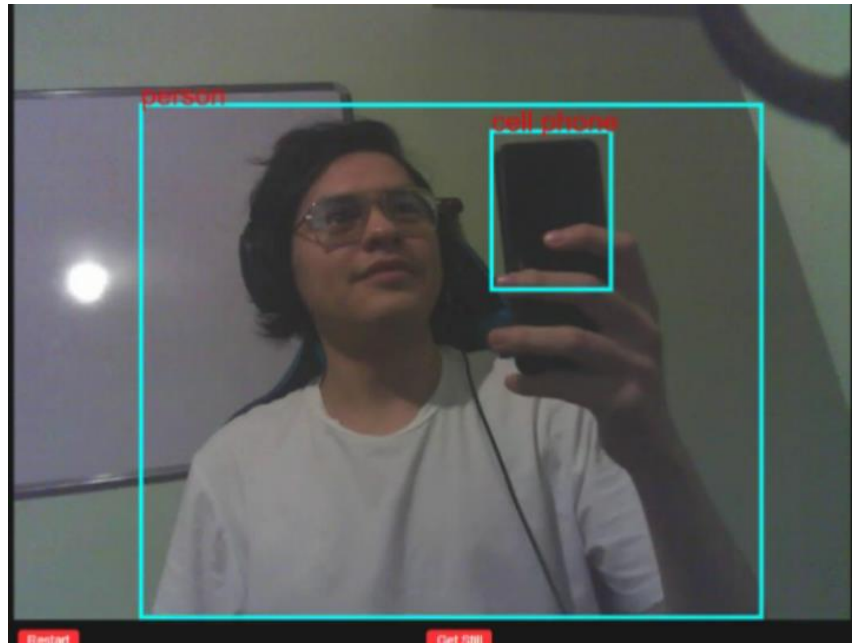
Tras dar click en el botón “get still”, obtenemos el video ya procesado por el modelo de AI



*Imagen 8. Observamos la detección de una persona y una silla además de la probabilidad de que sea ese objeto*

A continuación, algunas muestras más de la detección de otros objetos:





*Imagen 9 y 10. Detección de diversos objetos*

**NOTA:** Los datos en que muestran al detectar un objeto reconocido por la IA son 3:

1. La forma o zona donde se encuentra el objeto (Recuadro de color azul)
2. El nombre del objeto (De color rojo en la parte de arriba de cuadro o zona del objeto)

3. Las coordenadas donde se encuentra el objeto y la probabilidad de que sea dicho objeto

## **Conclusiones**

### ***Murrieta Villegas Alfonso***

En el presente proyecto aprendimos una de las mayores ventajas que nos pueden dar los microcontroladores que es la disponibilidad de crear proyectos de gran escalabilidad y alcance con un bajo costo, además y como es debido aplicamos diversos conceptos relacionados con varias materias:

Aplicamos conceptos de Microcomputadoras en la construcción y programación de las diversas funciones y algoritmos empleados para el tratamiento de los datos crudos obtenidos por una cámara integrada en el microcontrolador, a su vez , del manejo de los diversos puertos que este mismo tiene.

También aplicamos conceptos relacionados a Dispositivos electrónicos para el correcto manejo de nuestro microcontrolador tanto para descargar el código mediante un UART además de la alimentación y conexiones del mismo.

Aplicamos conceptos de desarrollo web y redes para el uso correcto de un web socket y de protocolos como HTTP para poder crear un pequeño sistema Cliente-Servidor.

Por último, aplicamos conceptos de Inteligencia Artificial para el manejo de un modelo de Inteligencia Artificial previamente entrenado por Google y que sirve sobre todo para consumirlo mediante un API con el objetivo de facilitar y ampliar el panorama de diversos proyectos.

### ***Reza Chavarria Sergio Gabriel***

A partir del uso del

***Valdespino Mendieta Joaquin***

En el presente proyecto