



## Práctica: Manipulación vía Wifi de un robot a través del ESP32

### Objetivo:

- Emplear un web-socket para la manipulación remota de un robot.
- Usar el ESP32-CAM para poder hacer streaming de video destinado a conocer el ambiente alrededor de un robot

### Introducción

La robótica es uno de los campos de mayor desarrollo dentro del mundo de la computación, sin duda, una de las ventajas que los microcontroladores nos ofrecen es que al ser dispositivos tan pequeños dan la posibilidad de poder emplearlos en un sinnúmero de actividades, en este caso, para la manipulación de robots de pocas acciones o capacidades.

Por otro lado, un puente H es un circuito electrónico que generalmente se usa para permitir a un motor eléctrico DC girar hacia adelante como hacia atrás.

### Descripción:

En prácticas previas comprendimos el patrón de diseño “Cliente - Servidor”, el cual está conformado principalmente de 2 entidades, la primera el **Servidor** que es la computadora o entidad encargada de desplegar el *Backend* o procesamiento de datos, por otro lado, el **Cliente** que es la computadora o dispositivo que solamente consume los datos o información que despliega el servidor.

En la presente práctica nuevamente se retomará esta arquitectura o patrón de diseño, con el objetivo de poder manipular un robot a través de un web-socket, además de que este mismo se empleará para poder hacer *streaming* de video con el propósito de conocer el ambiente que rodea a nuestro robot.

### Tabla de entradas y salidas:

A continuación, se muestra una imagen con las entradas y salidas del ESP32-CAM:

Tabla 1. Puertos a utilizar del ESP32

Puerto	Tipo	Descripción
GPIO 13	Output	Se usará para mandar el movimiento hacia la derecha
GPIO 14	Output	Se usará para mandar el movimiento hacia la atrás
GPIO 15	Output	Se usará para mandar el movimiento hacia la izquierda
GPIO 4	Output	Se usará para mandar el movimiento hacia la adelante
GPIO 2	Output	Este pin se usará para mandar el estado al led integrado en el ESP32

NOTA: Debido a que haremos uso de la cámara integrada del ESP32-CAM, es necesario también declarar sus respectivos pines, sin embargo, para eso tenemos 2 opciones, una es directamente hacer uso de la biblioteca que nos ofrece ESPRESSIF o como segunda opción agregar

manualmente cada puerto, en este caso sobre todo por optimización de espacio en memoria se decidió hacer explícito y manual la declaración de estos puertos: A continuación, una captura de pantalla de los puertos asociados a la cámara:

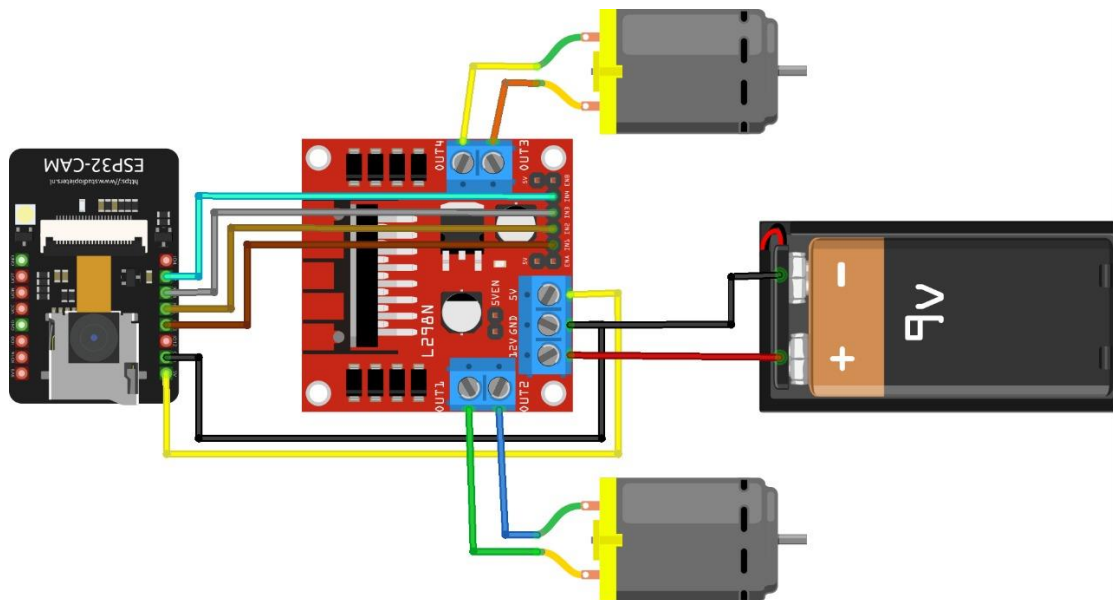
```
#if defined(CAMERA_MODEL_WROVER_KIT)
#define PWDN_GPIO_NUM    -1
#define RESET_GPIO_NUM   -1
#define XCLK_GPIO_NUM     21
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27

#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
```

Cabe destacar que es necesario mencionar el modelo específico de ESP32-CAM a utilizar en este caso el **wrover model kit**

### Diagrama de conexiones:

A continuación, se muestra el diagrama de conexiones, además de nuestro microcontrolador haremos uso del puente H L298 como recurso intermedio entre la conexión de nuestro microcontrolador y nuestros respectivos motores DC.



fritzing

Figura 1. Esquema de conexiones entre el ESP32 CAM y el puente H con sus respectivos servomotores



Cabe destacar que para poder hacer la descarga del programa se hace la forma que previamente se ha abordado, sin embargo, una vez cargado el programa en memoria, el ESP32-CAM se conectará y alimentará directamente del puente H.

### Listado del programa y descripción:

A continuación, se muestra paso a paso cada una de las partes que conforman la presente práctica, cabe destacar que debido a que este proyecto comprende una interfaz web, un web socket y además streaming de video, se separó el código en 3 principales archivos que serán explicados a continuación:

#### 1. Código principal

Denominado como ESP32CAM\_car, es el código principal del proyecto en Arduino y comprende de manera general los siguientes aspectos:

##### 1.1 Puertos y bibliotecas

Declaración de puertos e importación de las bibliotecas necesarias para hacer funcionar el proyecto, en este caso observamos que incluiremos la biblioteca estándar para el uso de la cámara del ESP32-CAM:

```
#include "esp_camera.h"
#include <WiFi.h>

//
// WARNING!!! Make sure that you have either selected ESP32 Wrover Module,
//           or another board which has PSRAM enabled
//
// Adafruit ESP32 Feather

#define CAMERA_MODEL_AI_THINKER

const char* ssid = "murryFly"; //Enter SSID WIFI Name
const char* password = "oQEeCponFG"; //Enter WIFI Password
```

Como bien se menciona en la tabla de entradas y salidas, se hará uso de algunas variables externas para el manejo y sobre todo para la manipulación del robot mediante el web-socket



```
#if defined(CAMERA_MODEL_WROVER_KIT)
#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

|
#else
#error "Camera model not selected"
#endif

// GPIO Setting
extern int gpLb = 2; // Left 1
extern int gpLf = 14; // Left 2
extern int gpRb = 15; // Right 1
extern int gpRf = 13; // Right 2
extern int gpLed = 4; // Light
extern String WiFiAddr = "";
```

## 1.2 Configuración de puertos y variables

Posteriormente en la función setup() se hará de forma explícita la declaración y configuración de cada uno de los puertos previamente instanciados, como se puede apreciar en la imagen inferior, lo primero que se hace es declarar cada uno de los puertos con sus respectivas variables de la manipulación del robot, posteriormente, se hace la configuración de cada uno de los puertos de la cámara.



```
void setup() {  
  Serial.begin(115200);  
  Serial.setDebugOutput(true);  
  Serial.println();  
  
  pinMode(gpLb, OUTPUT); //Left Backward  
  pinMode(gpLf, OUTPUT); //Left Forward  
  pinMode(gpRb, OUTPUT); //Right Forward  
  pinMode(gpRf, OUTPUT); //Right Backward  
  pinMode(gpLed, OUTPUT); //Light  
  
  //initialize  
  digitalWrite(gpLb, LOW);  
  digitalWrite(gpLf, LOW);  
  digitalWrite(gpRb, LOW);  
  digitalWrite(gpRf, LOW);  
  digitalWrite(gpLed, LOW);  
  
  camera_config_t config;  
  config.ledc_channel = LEDC_CHANNEL_0;  
  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_sscb_sda = SIOD_GPIO_NUM;  
  config.pin_sscb_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_JPEG;  
}
```

### 1.3 Lógica principal y cíclica del proyecto

Por último, a través de código genérico que podemos encontrar en el repositorio oficial del ESP32 (<https://github.com/espressif/arduino-esp32>) haremos la declaración y funcionamiento del streaming de video, es en esta parte donde se configuran aspectos como la calidad de la imagen además de los respectivos tamaños o formatos de transmisión de video. A su vez y como anteriormente se ha hecho, se mostrará información relevante a través del monitor serial, específicamente se desplegará el estado del web socket además de la respectiva ruta URL para poder manipular el robot.



```
//init with high specs to pre-allocate larger buffers
if(psramFound()){
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
}

// camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
}

//drop down frame size for higher initial frame rate
sensor_t * s = esp_camera_sensor_get();
s->set_framesize(s, FRAMESIZE_CIF);

WiFi.begin(ssid, password);

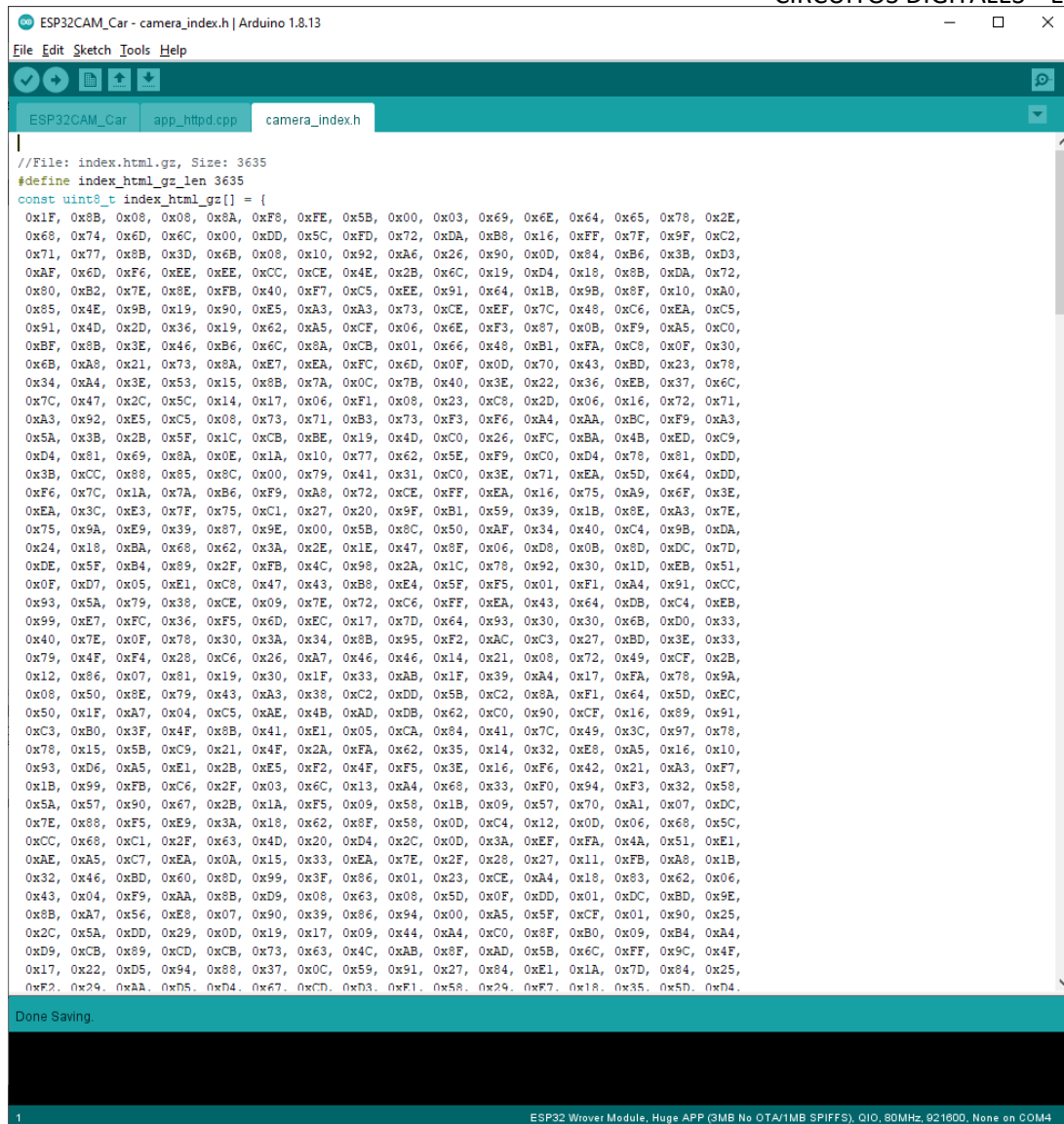
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");

startCameraServer();

Serial.print("Camera Ready! Use 'http://");
Serial.print(WiFi.localIP());
WiFiAddr = WiFi.localIP().toString();
Serial.println("' to connect");
}
```

## 2. Cámara\_index.h

Este programa realmente no es importante desde el punto de vista del desarrollo, realmente, es una biblioteca desarrollada y traducida a hexadecimal por ESPRESSIF con la finalidad de poder emplear el stream de video en el ESP32-CAM, sin embargo, es necesario incluirla en el proyecto debido a que esta será a su vez cargada al momento de compilar y enviar el programa completo al microcontrolador:



```

ESP32CAM_Car - camera_index.h | Arduino 1.8.13
File Edit Sketch Tools Help
ESP32CAM_Car app_httpd.cpp camera_index.h
//File: index.html.gz, Size: 3635
#define index_html_gz_len 3635
const uint8_t index_html_gz[] = {
0x1F, 0x8B, 0x08, 0x08, 0x8A, 0xF8, 0xFE, 0x5B, 0x00, 0x03, 0x69, 0x6E, 0x64, 0x65, 0x78, 0x2E,
0x68, 0x74, 0x6D, 0x6C, 0x00, 0xDD, 0x5C, 0xFD, 0x72, 0xDA, 0xB8, 0x16, 0xFF, 0x7F, 0x9F, 0xC2,
0x71, 0x77, 0x8B, 0x3D, 0xEB, 0x08, 0x10, 0x92, 0xA6, 0x26, 0x90, 0x0D, 0x84, 0xB6, 0x3B, 0xD3,
0xAF, 0x6D, 0xF6, 0xEE, 0xEE, 0xCC, 0xCE, 0x4E, 0x2B, 0x6C, 0x19, 0xD4, 0x18, 0xB8, 0xDA, 0x72,
0x80, 0xB2, 0x7E, 0x8E, 0xFB, 0x40, 0xF7, 0xC5, 0xEE, 0x91, 0x64, 0x1B, 0x9B, 0x8F, 0x10, 0xA0,
0x85, 0x4E, 0x9B, 0x19, 0x90, 0xE5, 0xA3, 0xA3, 0x73, 0xCE, 0xEF, 0x7C, 0x48, 0xC6, 0xEA, 0xC5,
0x91, 0x4D, 0x2D, 0x36, 0x19, 0x62, 0xA5, 0xCF, 0x06, 0x6E, 0xF3, 0x87, 0x0B, 0xF9, 0xA5, 0xC0,
0xBF, 0x8B, 0x3E, 0x46, 0xB6, 0x6C, 0x8A, 0xCB, 0x01, 0x66, 0x48, 0xB1, 0xFA, 0xC8, 0x0F, 0x30,
0x6B, 0xA8, 0x21, 0x73, 0x8A, 0xE7, 0xEA, 0xFC, 0x6D, 0x0F, 0x0D, 0x70, 0x43, 0xBD, 0x23, 0x78,
0x34, 0xA4, 0x3E, 0x53, 0x15, 0x8B, 0x7A, 0x0C, 0x7B, 0x40, 0x3E, 0x22, 0x36, 0xEB, 0x37, 0x6C,
0x7C, 0x47, 0x2C, 0x5C, 0x14, 0x17, 0x06, 0xF1, 0x08, 0x23, 0xC8, 0x2D, 0x06, 0x16, 0x72, 0x71,
0xA3, 0x92, 0xE5, 0xC5, 0x08, 0x73, 0x71, 0xB3, 0x73, 0xF3, 0xF6, 0xA4, 0xAA, 0xBC, 0xF9, 0xA3,
0x5A, 0x3B, 0x2B, 0x5F, 0x1C, 0xCB, 0xBE, 0x19, 0x4D, 0xC0, 0x26, 0xFC, 0xBA, 0x4B, 0xED, 0xC9,
0xD4, 0x81, 0x69, 0x8A, 0x0E, 0x1A, 0x10, 0x77, 0x62, 0x5E, 0xF9, 0xC0, 0xD4, 0x78, 0x81, 0xDD,
0x3B, 0xCC, 0x88, 0x85, 0x8C, 0x00, 0x79, 0x41, 0x31, 0xC0, 0x3E, 0x71, 0xEA, 0x5D, 0x64, 0xDD,
0xF6, 0x7C, 0x1A, 0x7A, 0xB6, 0xF9, 0xA8, 0x72, 0xCE, 0xFF, 0xEA, 0x16, 0x75, 0xA9, 0x6F, 0x3E,
0xEA, 0x3C, 0x33, 0x7F, 0x75, 0xC1, 0x27, 0x20, 0x9F, 0xB1, 0x59, 0x39, 0x1B, 0x8E, 0xA3, 0x7E,
0x75, 0x9A, 0xE9, 0x39, 0x87, 0x9E, 0x00, 0x5B, 0x8C, 0x50, 0xAF, 0x34, 0x40, 0xC4, 0x9B, 0xDA,
0x24, 0x18, 0xBA, 0x68, 0x62, 0x3A, 0x2E, 0x1E, 0x47, 0x8F, 0x06, 0xD8, 0x0B, 0x8D, 0xDC, 0x7D,
0xDE, 0x5F, 0xB4, 0x89, 0x2F, 0xFB, 0x4C, 0x98, 0x2A, 0x1C, 0x78, 0x92, 0x30, 0x1D, 0xEB, 0x51,
0x0F, 0xD7, 0x05, 0xE1, 0xC8, 0x47, 0x43, 0xB8, 0xEA, 0x5F, 0xF5, 0x01, 0xF1, 0xA4, 0x91, 0xCC,
0x93, 0x5A, 0x79, 0x38, 0xCE, 0x09, 0x7E, 0x72, 0xC6, 0xFF, 0xEA, 0x43, 0x64, 0xDB, 0xC4, 0xEB,
0x99, 0xE7, 0xFC, 0x36, 0xF5, 0x6D, 0xEC, 0x17, 0x7D, 0x64, 0x93, 0x30, 0x30, 0x6B, 0xD0, 0x33,
0x40, 0x7E, 0x0F, 0x78, 0x30, 0x3A, 0x34, 0x8B, 0x95, 0xF2, 0xAC, 0xC3, 0x27, 0xBD, 0x3E, 0x33,
0x79, 0x4F, 0xF4, 0x28, 0xC6, 0x26, 0xA7, 0x46, 0x46, 0x14, 0x21, 0x08, 0x72, 0x49, 0xCF, 0x2B,
0x12, 0x86, 0x07, 0x81, 0x19, 0x30, 0x1F, 0x33, 0xAB, 0x1F, 0x39, 0xA4, 0x17, 0xFA, 0x78, 0x9A,
0x08, 0x50, 0x8E, 0x79, 0x43, 0xA3, 0x38, 0xC2, 0xDD, 0x5B, 0xC2, 0x8A, 0xF1, 0x64, 0x5D, 0xEC,
0x50, 0x1F, 0xA7, 0x04, 0xC5, 0xAE, 0x4B, 0xAD, 0xDB, 0x62, 0xC0, 0x90, 0xCF, 0x16, 0x89, 0x91,
0xC3, 0xB0, 0x3F, 0x4F, 0x8B, 0x41, 0xE1, 0x05, 0xCA, 0x84, 0x41, 0x7C, 0x49, 0x3C, 0x97, 0x78,
0x78, 0x15, 0x5B, 0xC9, 0x21, 0x4F, 0x2A, 0xFA, 0x62, 0x35, 0x14, 0x32, 0xE8, 0xA5, 0x16, 0x10,
0x93, 0xD6, 0xA5, 0xE1, 0x2B, 0xE5, 0xF2, 0x4F, 0xF5, 0x3E, 0x16, 0xF6, 0x42, 0x21, 0xA3, 0xF7,
0x1B, 0x99, 0xFB, 0xC6, 0x2F, 0x03, 0x6C, 0x13, 0xA4, 0x68, 0x33, 0xF0, 0x94, 0xF3, 0x32, 0x58,
0x5A, 0x57, 0x90, 0x67, 0x2B, 0x1A, 0xF5, 0x09, 0x58, 0x1B, 0x09, 0x57, 0x70, 0xA1, 0x07, 0xDC,
0x7E, 0x88, 0xF5, 0xF9, 0x3A, 0x18, 0x62, 0x8F, 0x58, 0x0D, 0xC4, 0x12, 0x0D, 0x06, 0x68, 0x5C,
0xCC, 0x68, 0xC1, 0x2F, 0x63, 0x4D, 0x20, 0xD4, 0x2C, 0x0D, 0x3A, 0xEF, 0xFA, 0x4A, 0x51, 0xE1,
0xAE, 0xA5, 0xC7, 0xEA, 0x0A, 0x15, 0x33, 0xEA, 0x7E, 0x2F, 0x28, 0x27, 0x11, 0xFB, 0xA8, 0x1B,
0x32, 0x46, 0xBD, 0x60, 0x8D, 0x99, 0x3F, 0x86, 0x01, 0x23, 0xCE, 0xA4, 0x18, 0x83, 0x62, 0x06,
0x43, 0x04, 0xF9, 0xAA, 0x8B, 0xD9, 0x08, 0x63, 0x08, 0x5D, 0x0F, 0xDD, 0x01, 0xDC, 0xBD, 0x9E,
0x8B, 0xA7, 0x56, 0xE8, 0x07, 0x90, 0x39, 0x86, 0x94, 0x00, 0xA5, 0x5F, 0xCF, 0x01, 0x90, 0x25,
0x2C, 0x5A, 0xDD, 0x29, 0x0D, 0x19, 0x17, 0x09, 0x44, 0xA4, 0xC0, 0x8F, 0xB0, 0x09, 0xB4, 0xA4,
0xD9, 0xCB, 0x89, 0xCD, 0xCB, 0x73, 0x63, 0x4C, 0xAB, 0x8F, 0xAD, 0x5B, 0x6C, 0xFF, 0x9C, 0x4F,
0x17, 0x22, 0xD5, 0x94, 0x88, 0x37, 0x0C, 0x59, 0x91, 0x27, 0x84, 0xE1, 0x1A, 0x7D, 0x84, 0x25,
0xF2, 0x29, 0xA2, 0x75, 0x74, 0x67, 0xC0, 0x73, 0xF1, 0x58, 0x29, 0xF7, 0x18, 0x35, 0x5D, 0x74.
}
Done Saving.
ESP32 Wrover Module, Huge APP (3MB No OTA/1MB SPIFFS), QIO, 80MHz, 921600, None on COM4

```

Figura 2. Código en binario del arreglo incluido dentro de la biblioteca para el manejo de la cámara del ESP32-CAM

### 3. app\_httpd.cpp

Es el código encargado de 3 tareas en general, la primera crear la interfaz web, la segunda conectar la interfaz web con el web socket para poder realizar la manipulación remota del robot y la tercera, la conexión del stream de video en la interfaz web

#### 3.1 Definición de variables y de estructuras de datos

Lo primero que debemos hacer hincapié es que debido a que el desarrollo de este proyecto es mediante un lenguaje que no es orientado a objetos hace que para poder crear entidades más complejas se creen lo que se denomina como “Estructuras de Datos”, específicamente en este caso se crearon 2 estructuras de datos en particular, la primera es un filtro para la transmisión de datos y la otra para el buffering de imágenes al momento de hacer el stream de video





```
#include "esp_http_server.h"
#include "esp_timer.h"
#include "esp_camera.h"
#include "img_converters.h"
#include "camera_index.h"
#include "Arduino.h"

extern int gpLb;
extern int gpLf;
extern int gpRb;
extern int gpRf;
extern int gpLed;
extern String WiFiAddr;

void WheelAct(int nLf, int nLb, int nRf, int nRb);

typedef struct {
    size_t size; //number of values used for filtering
    size_t index; //current value index
    size_t count; //value count
    int sum;
    int * values; //array to be filled with values
} ra_filter_t;

typedef struct {
    httpd_req_t *req;
    size_t len;
} jpg_chunking_t;
```

### 3.2 Apartado de framing y buffering de video

Para este apartado nuevamente se uso código base del repositorio oficial de ESPRESSIF, esto debido a que el propósito de la práctica no es crear software que maneje buffering o freaming de video, sino el emplear ya directamente el stream resultante como recurso para la manipulación de un robot





```
static ra_filter_t ra_filter;
httpd_handle_t stream_httpd = NULL;
httpd_handle_t camera_httpd = NULL;

static ra_filter_t * ra_filter_init(ra_filter_t * filter, size_t sample_size){
    memset(filter, 0, sizeof(ra_filter_t));

    filter->values = (int *)malloc(sample_size * sizeof(int));
    if(!filter->values){
        return NULL;
    }
    memset(filter->values, 0, sample_size * sizeof(int));

    filter->size = sample_size;
    return filter;
}

static int ra_filter_run(ra_filter_t * filter, int value){
    if(!filter->values){
        return value;
    }
    filter->sum -= filter->values[filter->index];
    filter->values[filter->index] = value;
    filter->sum += filter->values[filter->index];
    filter->index++;
    filter->index = filter->index % filter->size;
    if (filter->count < filter->size) {
        filter->count++;
    }
    return filter->sum / filter->count;
}

static size_t jpg_encode_stream(void * arg, size_t index, const void* data, size_t len){
    jpg_chunking_t *j = (jpg_chunking_t *)arg;
    if(!index){
        j->len = 0;
    }
    if(httpd_resp_send_chunk(j->req, (const char *)data, len) != ESP_OK){
        return 0;
    }
    j->len += len;
    return len;
}
```

### 3.3 Funciones estáticas para el web-socket y el stream de video

Para este apartado solamente se mencionarán las funciones bases que se emplearon para el manejo de video:



```
static size_t jpg_encode_stream(void * arg, size_t index, const void* data, size_t len){
    jpg_chunking_t *j = (jpg_chunking_t *)arg;
    if(!index){
        j->len = 0;
    }
    if(httpd_resp_send_chunk(j->req, (const char *)data, len) != ESP_OK){
        return 0;
    }
    j->len += len;
    return len;
}

static esp_err_t capture_handler(httpd_req_t *req){
    camera_fb_t * fb = NULL;
    esp_err_t res = ESP_OK;
    int64_t fr_start = esp_timer_get_time();

    fb = esp_camera_fb_get();
    if (!fb) {
        Serial.printf("Camera capture failed");
        httpd_resp_send_500(req);
        return ESP_FAIL;
    }

    httpd_resp_set_type(req, "image/jpeg");
    httpd_resp_set_hdr(req, "Content-Disposition", "inline; filename=capture.jpg");
    ...

static esp_err_t status_handler(httpd_req_t *req){
    static char json_response[1024];

    sensor_t * s = esp_camera_sensor_get();
    char * p = json_response;
    *p++ = '{';
```

### 3.4 Interfaz Web

Al igual que se hizo en la práctica del web-socket, a través de código en C es como se desarrollo la interfaz web, la diferencia entre ambas prácticas, es que en este caso se manejo en una función por separado:

### 3.5 Setter o actuadores de variables

A pesar de que C no es un lenguaje orientado a objetos, el paradigma puede ser empleado para poder aterrizar lógicamente ciertos procedimientos o abstracciones, en este caso se desarrollo una función o método actuador también conocidos como setter que se encargarán de cambiar los estados de cada una de las variables del robot, en este caso la de movimiento hacia adelante, atrás, izquierda, derecha y la de prendido y apagado del LED.



```
static esp_err_t go_handler(httpd_req_t *req){
    WheelAct(HIGH, LOW, HIGH, LOW);
    Serial.println("Go");
    httpd_resp_set_type(req, "text/html");
    return httpd_resp_send(req, "OK", 2);
}

static esp_err_t back_handler(httpd_req_t *req){
    WheelAct(LOW, HIGH, LOW, HIGH);
    Serial.println("Back");
    httpd_resp_set_type(req, "text/html");
    return httpd_resp_send(req, "OK", 2);
}

static esp_err_t left_handler(httpd_req_t *req){
    WheelAct(HIGH, LOW, LOW, HIGH);
    Serial.println("Left");
    httpd_resp_set_type(req, "text/html");
    return httpd_resp_send(req, "OK", 2);
}

static esp_err_t right_handler(httpd_req_t *req){
    WheelAct(LOW, HIGH, HIGH, LOW);
    Serial.println("Right");
    httpd_resp_set_type(req, "text/html");
    return httpd_resp_send(req, "OK", 2);
}

static esp_err_t stop_handler(httpd_req_t *req){
    WheelAct(LOW, LOW, LOW, LOW);
    Serial.println("Stop");
    httpd_resp_set_type(req, "text/html");
    return httpd_resp_send(req, "OK", 2);
}

static esp_err_t ledon_handler(httpd_req_t *req){
    digitalWrite(gpLed, HIGH);
    Serial.println("LED ON");
    httpd_resp_set_type(req, "text/html");
    return httpd_resp_send(req, "OK", 2);
}

static esp_err_t ledoff_handler(httpd_req_t *req){
    digitalWrite(gpLed, LOW);
    Serial.println("LED OFF");
    httpd_resp_set_type(req, "text/html");
    return httpd_resp_send(req, "OK", 2);
}
```

### 3.6 Función principal del código

Como se puede observar en la última parte del código principal del proyecto, lo único que se hace llamado de este último código es la función **startCameraServer()**; que es la encargada de realizar el llamado de los anteriores setter además de las funciones de stream de video en la interfaz web:



```
void startCameraServer(){
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();

    httpd_uri_t go_uri = {
        .uri      = "/go",
        .method    = HTTP_GET,
        .handler    = go_handler,
        .user_ctx  = NULL
    };

    httpd_uri_t back_uri = {
        .uri      = "/back",
        .method    = HTTP_GET,
        .handler    = back_handler,
        .user_ctx  = NULL
    };

    httpd_uri_t stop_uri = {
        .uri      = "/stop",
        .method    = HTTP_GET,
        .handler    = stop_handler,
        .user_ctx  = NULL
    };

    httpd_uri_t left_uri = {
        .uri      = "/left",
        .method    = HTTP_GET,
        .handler    = left_handler,
        .user_ctx  = NULL
    };

    httpd_uri_t right_uri = {
        .uri      = "/right",
        .method    = HTTP_GET,
        .handler    = right_handler,
        .user_ctx  = NULL
    };
};
```

### Interfaz y funcionamiento:

A continuación, una imagen general de la interfaz web obtenido tras la compilación y descarga de nuestro código:

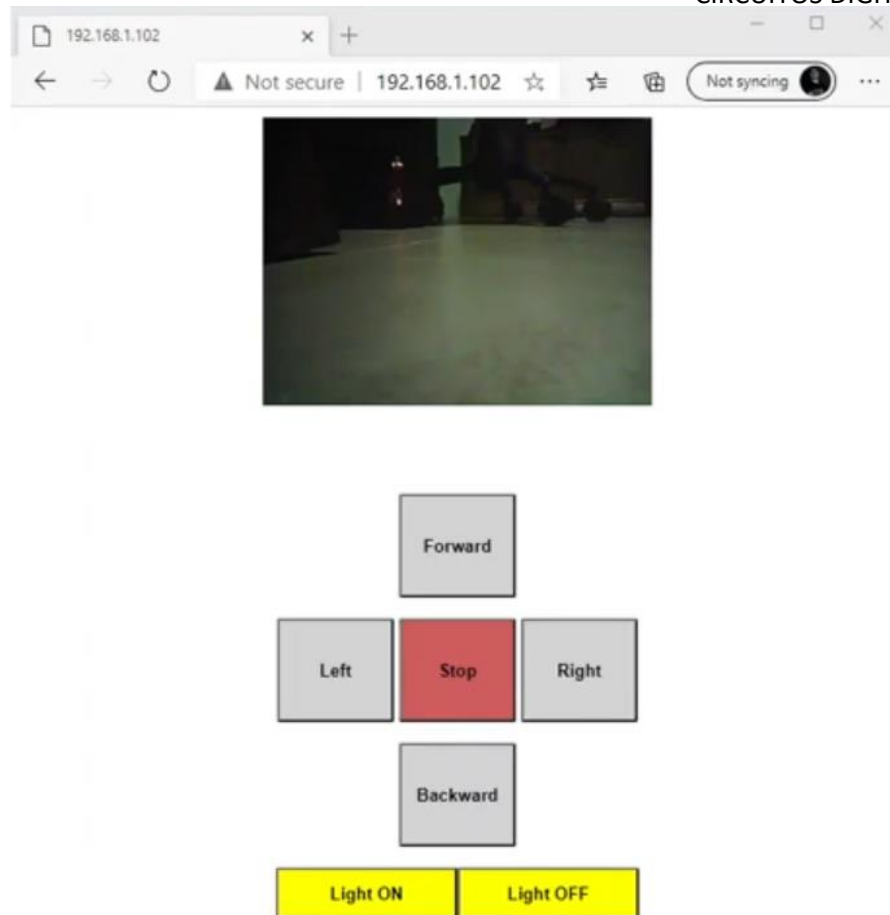


Figura 3. Interfaz web para el manejo y teleoperación del robot, se observan los botones de movimiento, botones para prender el led integrado en el ESP32 además del stream de video

Por otro lado, también se comparte una imagen global del funcionamiento del robot y de cómo es que se manipularía a través de esta interfaz (Para más detalles del funcionamiento, revisar el video adjunto en la sección de recursos extras)

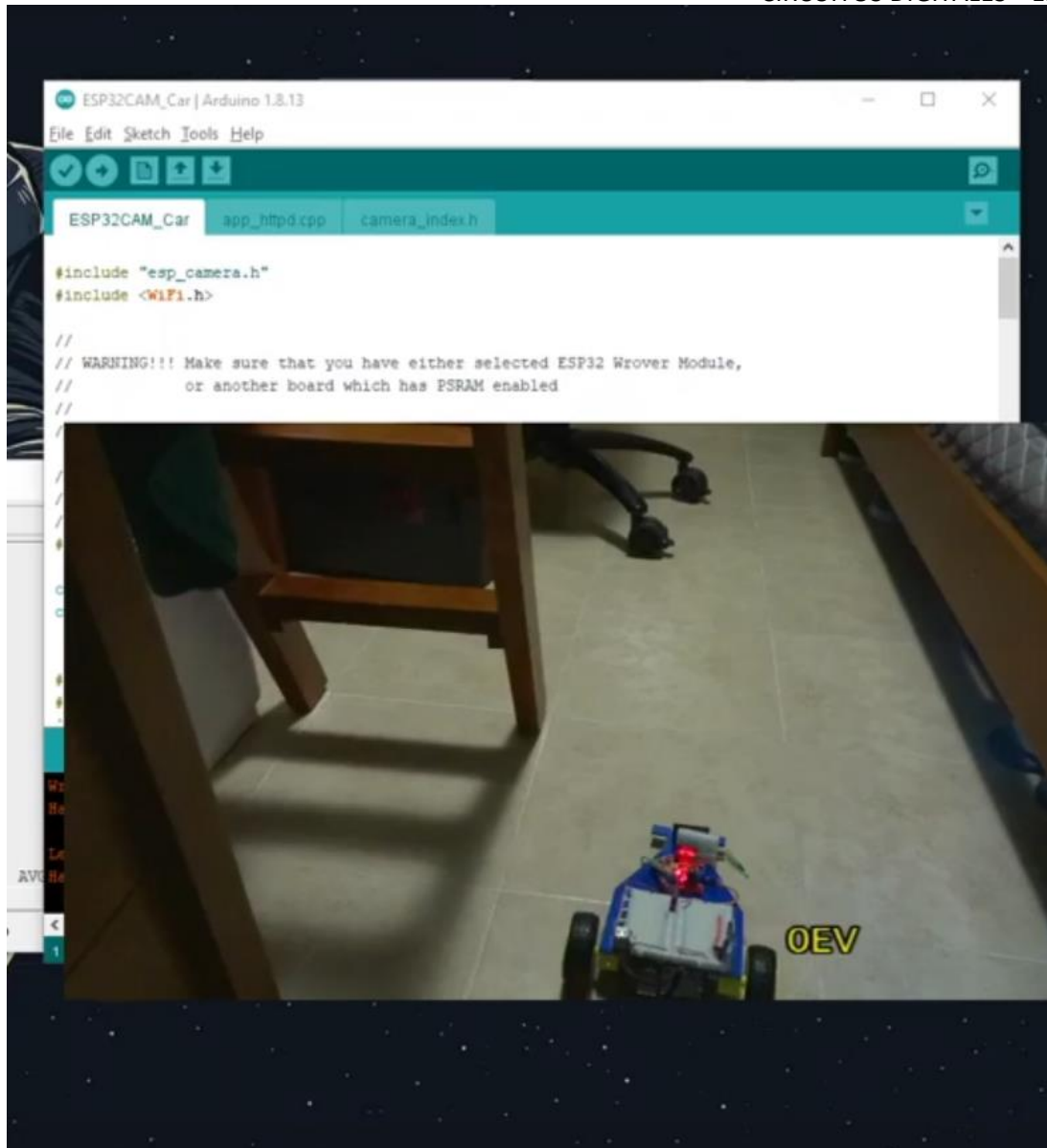


Figura 4. Movimiento del robot a través del web socket y empleando la interfaz web anterior

### Recursos extras:

Video de funcionamiento de la presente práctica:

<https://youtu.be/aVfBxol9CI8>