Escuela Universitaria
Politécnica - La Almunia
Centro adscrito
Universidad Zaragoza

20/06/2022

Instrumentación electrónica avanzada

DETECCIÓN DE COLOR

ESP32-CAM
LABVIEW
UDP

PROFESOR **DR.** DAVID ASIAIN ANSORENA
AUTOR NAOUFAL EL RHAZZALI
EUPLA **ING**. MECATRÓNICA

Contenido

Figuras

Al Prof. David Asiain

Agradezco al Prof. David Asiain por brindarme la oportunidad de participar en esta asignatura tan divertida y llena de píldoras de aprendizaje. Gracias por dejarme asistir y por toda la ayuda y la paciencia.

Naoufal El Rhazzali

Resumen

Esta aplicación emplea el ESP32-CAM para la detección de color, para posteriormente aumentar un contador que alcanza un valor máximo. Esto último conduce en que LabVIEW envía un mensaje al ESP32-CAM con una orden de encender el Led interno de la misma, así mismo, reinicia el contador.

Abstract

This application uses ESP32-CAM to color detection. Each detection increases a counter until reach a maximum value. Once that happens LabVIEW sends an order to the ESP32-CAM to turn its internal LED on, likewise the counter is rested.

Marco teórico

LabVIEW

En palabras del fabricante; "*LabVIEW es un entorno de programación gráfica que los ingenieros utilizan para desarrollar sistemas de pruebas automatizados de investigación, validación y producción"*. [1]

Este entorno de desarrollo (LabVIEW) se basa en la programación gráfica, la cual constituye un paradigma distinto al habitual paradigma basado en "instrucciones" escritas en texto. En la literatura del fabricante se compara el paso de la comunidad científica después de los años 50s a programar en FORTRAN (lenguaje creado por IBM) en vez de usar ensamblador y, con el paso de los ingenieros a utilizar el lenguaje G (lenguaje gráfico de LabVIEW) como mejor alternativa a otros lenguajes basados en texto. Se señala que, al graficar la programación se sube un escalón más en la abstracción y con ello se puede implementar aplicaciones más potentes, dando facilidad al desarrollador para llevar a cabo ideas que, con "lenguajes textuales" sería una tarea tediosa.

El hecho de ser G un lenguaje gráfico, no lo priva de las características fundamentales de un lenguaje de programación (tipos de datos, bucles, variables, eventos). Así mismo es posible programar con orientación a procesos como a objetos. No obstante, cabe señalar una diferencia muy importante, y quizás es la clave de poder generar soluciones muy potentes con este tipo de lenguaje (como Agilent VEE, Microsoft Visual Programming Language, Apple Quartz Composer, etc.), se refiere a la orientación a datos, si la expresión lo permite. La orientación a datos consiste en gobernar el flujo de ejecución por el flujo de datos. El viaje de los datos de un nodo a otro es el que determina el flujo de ejecución, y esto ya es una diferencia importante respecto a la programación secuencial de los típicos lenguajes de programación textual, C/C++, Java, Python, Matlab, R, etc.

¿Por qué LabVIEW?

1. LabVIEW es una herramienta estándar en la industria. El manejo y la familiarización con LabVIEW agrega un valor añadido al futuro ingeniero en su Currículum Vitae.

2. LabVIEW proporciona una integración con hardware y más de 1,000 funciones integradas para adquisición de datos y análisis.

Finalmente, aprovechando este paradigma, se desarrollará una aplicación de adquisición de imágenes generadas por medio de una ESP32-CAM, para más tarde postprocesar y generar el valor añadido. [2]

Ai-Thinker ESP32-CAM

Figura 1 Placa Ai-Thinker ESP32-CAM [3]

Se ha trabajado en este proyecto con el módulo ESP32-CAM, el cual es el último módulo de cámara de tamaño pequeño lanzado por Essence. Este módulo tienes unas dimensiones de 27.40x5.4x5 (mm) y alcanza una corriente de sueño profundo de hasta los 6 mA. Así mismo cuenta con conexión Wifi (802,11b/g/n) Y Bluetooth BT/BLE Soc. El encapsulado es de tipo DIP. [4]

Características

- CPU de 32 bits de doble núcleo de baja potencia.
- Frecuencia principal de hasta 240 MHz, potencia de cálculo de hasta 600 DMIPS
- SRAM integrada de 520 KB, PSRAM externa de 8 MB
- Admite UART/SPI/I^{2}C/PWM/ADC/DAC y otras interfaces.
- Admite cámaras OV2640 y OV7670, con flash incorporado.
- Soporte de carga de imagen Wifi.
- Soporte de tarjeta TF.
- Admite múltiples modos de suspensión.
- LWIP integrado y FreeRTOS.
- Admite el modo de trabajo STA/AP/STA+AP
- Admite la configuración de red con un solo clic de Smart Config/AirKiss

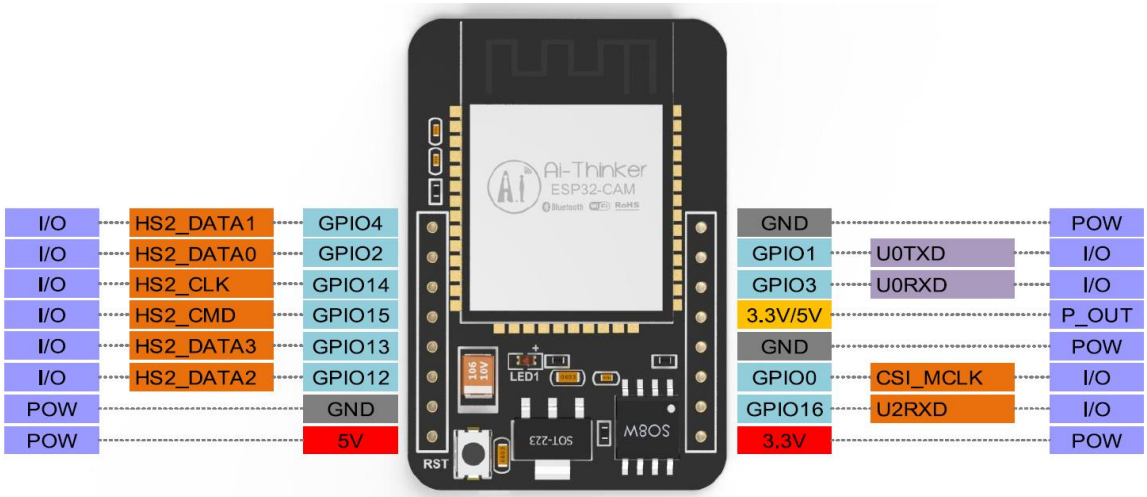

Figura 2 Pinout Ai-Thinker ESP32-CAM [4]

Observaciones para uso de cámara

- Asegurarse de que el voltaje de entrada del módulo sea de al menos de lo contrario, la imagen puede tener líneas de agua.
- El pin GPIO32 controla la alimentación de la cámara. Cuando la cámara esté funcionando, se ha de poner a GND el GPIO32.
- Dado que IO_0 está conectado al XCLK de la cámara, hay que dejarlo al aire cuando no conectarlo a un nivel alto o bajo.

Comunicación Udp

Udp es un protocolo de comunicación de la capa de transporte, no es orientado a conexión. Este protocolo sólo envía paquetes entre aplicaciones, y deja que las aplicaciones construyan sus propios protocolos en la parte superior según sea necesario. [5]

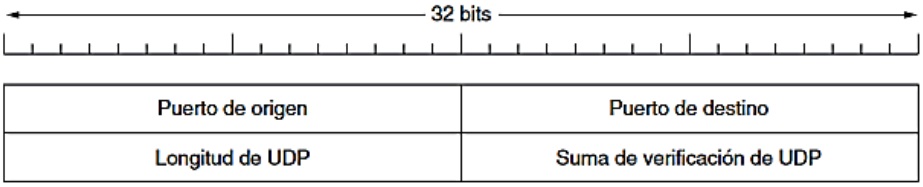

Figura 3 El encabezado UDP. [5]

Desarrollo

El primer paso en el desarrollo de esta aplicación es la adquisición de imágenes mediante la Esp32-CAM. Para ello se ha escrito el siguiente código en C++ en Arduino.

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

#define LED                       4
#define CHUNK_LENGTH          1460

camera_fb_t* fb = NULL;
uint8_t encodedBuffer[20000];

boolean connected = false;
const char* ssid = "XXXXXX";
const char* password = " XXXXXX ";

WiFiUDP udp;
const char* udpAddress = "XXX.XXX.XXX.XXX";
const int udpPort = XXXX;

void connectToWiFi(const char* ssid, const char* pwd);
void camaraEvent();
void takeImag();
uint32_t encode(const uint8_t* buffer,
                uint32_t size,
                uint8_t* encodedBuffer);
void sendPacketData(const char* buf,
                    uint16_t len,
                    uint16_t chunkLength);
void ledControl();

void setup() {
  Serial.begin(115200);
  pinMode(LED, OUTPUT);

  connectToWiFi(ssid, password);
  camaraEvent();
}

void loop() {
  if (connected) {
    takeImag();
    uint16_t encode_len = encode(fb->buf, fb->len, encodedBuffer);
    sendPacketData((const char*)encodedBuffer,
                   encode_len, CHUNK_LENGTH);

    esp_camera_fb_return(fb);
  }
  ledControl();
}
```

En las primeras líneas de código se encuentran los “includes”, estos incluyen las librerías que facilitan la programación y configuración del Wifi, la conexión udp y la configuración e interacción con la cámara que trae el ESP32-CAM.

A continuación, le siguen las definiciones. Se definen el LED que es el pin del Led interno del ESP32-CAM y la longitud de paquete según el troceo que se hace en el envío de la imagen.

En adición, cabe mencionar las estructuras de datos relativas a la imagen. Así, se declaran un puntero a estructura que contiene toda la información de la imagen (fb) y un buffer donde se almacenarán los bytes codificados. Luego se definen una serie de variables relativas a la conexión, tanto ***connected***, ***ssid*** como ***password*** tienen que ver con la comunicación Wifi. Por otra parte, ***udpAddress***, ***udpPort*** son constantes relacionadas con el protocolo udp, así mismo el objeto que se instancia de la clase WiFiUDP.

Finalmente, se declaran las firmas de las funciones que su uso queda explícito por el propio nombre de las mismas.

El problema que se ha encontrado en la adquisición de imágenes y su envío a LabVIEW fue la sincronización. Ocurre, que los bytes de imagen se envían troceado en paquetes de 1460 Bytes. Esto último impide la posibilidad un envío continuo de imágenes para poder conseguir una transmisión en vivo. El problema de la sincronización hace que no se sepa desde LabVIEW cuando empieza una imagen y cuando acaba, resultando en una corrupta escritura de imágenes sobre archivos binarios.

Al hilo de lo anterior, la solución que se plantea es un algoritmo que implementa un protocolo SLIP (Serial Line Internet Protocol). Este protocolo resuelve el problema de sincronización al codificar los paquetes con un byte de inicio de paquete utilizando el 192. Así entre al enviar una foto y la que le sigue, las dos empezarán con el 192, por ello el algoritmo de decodificación en LabVIEW detectará que identificará íntegramente una imagen entre cada 192.

No obstante, lo anterior nos genera un problema; ¿Qué pasa si aparece un 192 en mitad de la imagen? ¿Se cerraría la imagen a mitad de decodificación en LabVIEW, generando un resultado corrompido? Para resolver este problema, se declara un ***enum*** como el siguiente.

```
enum    {       END = 192,
                ESC = 219,
                ESC_END = 220,
                ESC_ESC = 221
        };
```

Como se ve, se han declarado 4 casos. Ello se resuelve el problema de encontrarse el byte especial de inicio 192. Ello lo hace, codificando dicho byte original de los bytes de la imagen como dos bytes seguidos el primero ESC (219) y el siguiente ESC_END (220). Esto último se decodificaría como sigue: Al encontrarse un ESC, se comprueba que el siguiente byte es un ESC_END, y si es así esos dos bytes se sustituyen por el END. No obstante, qué pasaría si se encontrase con un ESC en los bytes originales de la imagen como lo codificamos sin confundirlos con los bytes especiales de codificación. Para resolver esto último, se emplea el último byte especial que ESC_ESC, tal que cuando se encuentra en los bytes originales de la imagen un ESC, este se sustituye por un byte ESC seguido de un ESC_ESC. Así en LabVIEW al encontrarse un ESC, se comprueba que lo que sigue es un ESC_ESC, y si es así dichos dos bytes serán decodificados como el byte ESC. Todo lo comentado en este párrafo, se ha implementado en C++ por el ***Prof. Dr. David Asiain***, como sigue:

```
uint32_t encode(const uint8_t* buffer,
                uint32_t size,
                uint8_t* encodedBuffer) {
        if (size == 0)return 0;

        size_t read_index  = 0; size_t write_index = 0;

        encodedBuffer[write_index++] = END;

        while (read_index < size) {
            if(buffer[read_index] == END)
                encodedBuffer[write_index++] = ESC;
                encodedBuffer[write_index++] = ESC_END;
                read_index++;
              }
```

```
            else if(buffer[read_index] == ESC) {
                encodedBuffer[write_index++] = ESC;
                encodedBuffer[write_index++] = ESC_ESC;
                read_index++;
              }
            else{
                encodedBuffer[write_index++] = buffer[read_index++];
              }
        }
        return write_index;
}
```

Resuelto problema de sincronización lo que queda es enviar mediante los métodos que nos brinda la clase WiFiUDP, el paquete respetando el troceo del mismo. Para ello, se ha empleado el siguiente algoritmo.

```
void sendPacketData(const char* buf,
                    uint16_t len,
                    uint16_t chunkLength) {

  uint8_t buffer[chunkLength];
  size_t blen = sizeof(buffer);
  size_t rest = len % blen;

  for (uint8_t i = 0; i < len / blen; ++i) {
    memcpy(buffer, buf + (i * blen), blen);
    udp.beginPacket(udpAddress, udpPort);
    udp.write(buffer, chunkLength);
    udp.endPacket();
  }

  if (rest) {
    memcpy(buffer, buf + (len - rest), rest);
    udp.beginPacket(udpAddress, udpPort);
    udp.write(buffer, rest);
    udp.endPacket();
  }
}
```

No hay nada especial acerca del algoritmo de envío, salvo el troceo de datos en paquetes de 1460 bytes. No obstante, siempre quedaría un paquete menor de 1460. Entonces se resuelve eso conociendo el resto al dividir el tamaño de array completo sobre 1460.

El siguiente paso es recibir los datos en LabVIEW, extraerlos y decodificarlos.

Packet extractor

Este módulo es un for While Loop que termina al ser la longitud del paquete nula.

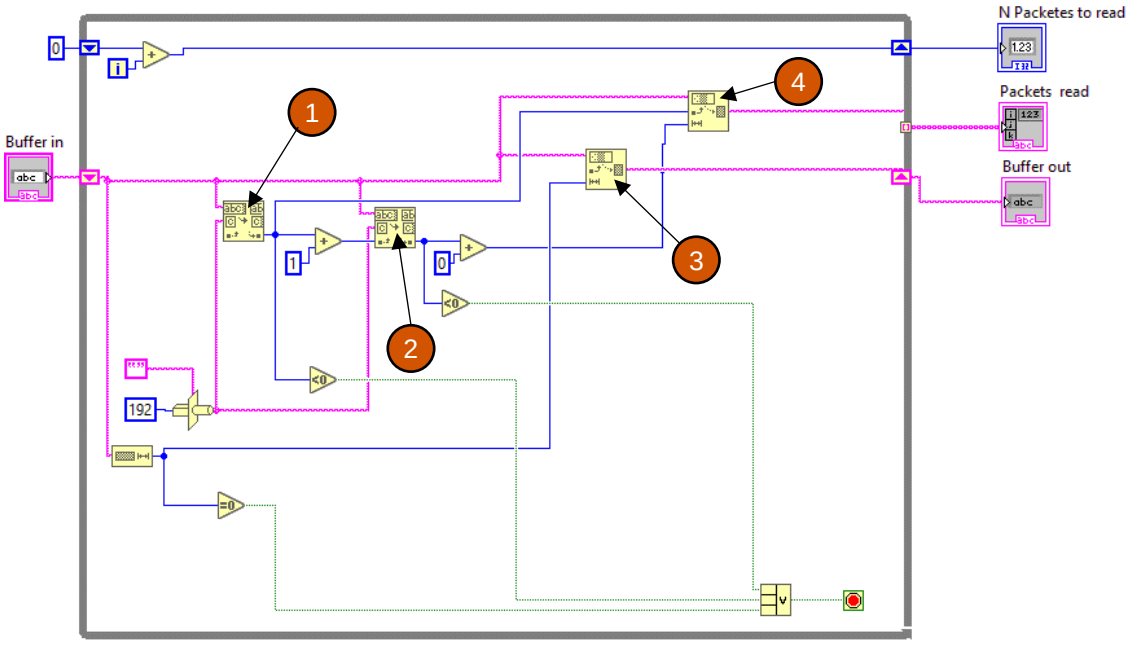

Figura 4 Modulo extractor de los paquetes enviados

La función señalada como (1) y (2) corresponden con lo que sigue.

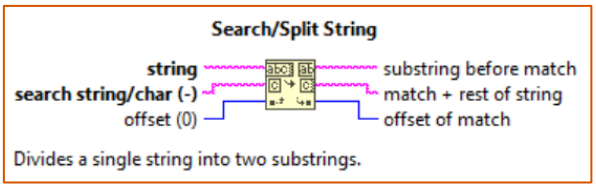

Figura 5 Search/Split String

La función señalada como (3) y (4) corresponden con lo que sigue.

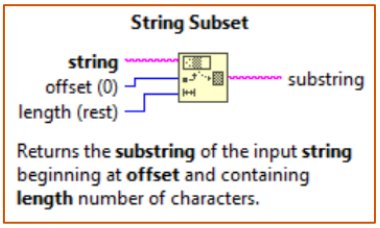

Figura 6 String Subset

La idea es que este extractor localizaría entre todos los paquetes que llegan una imagen íntegra. Hágase notar que la función (1) recibe la cadena de String leída desde el udp, el siguiente argumento llega desde la salida de un ***typeCast*** que convierte el 192 (END) a tipo ***char***. Así lo que haría es devolver a la salida cableada el índice donde encontró el 192. Eso, indica el inicio de una imagen. Ese índice + 1 se lo pasa a la función (2) que vuelve a recibir la misma cadena de String recibida del udp, así mismo el carácter que debe buscar. No obstante, recibe un offset de la primera función (1). Esto último, hace que la búsqueda se haga a partir del siguiente carácter dónde se localizó el byte de inicio. Al final acaba devolviendo un segundo índice de END. Esos dos índices los recibe la función (4) que extrae un paquete iniciado y finalizado por END. La función (3), devuelve el String restante, que se vuelve a someterse a la misma operación

gracias a la existencia de un registro de desplazamiento a izquierdas. Así hasta finalizar todo el String. En esta fase, queda decodificar según las reglas definidas en el ***enum*** de arriba.

Para la decodificación se recurre al módulo de Packets Decoder.

SLIP Decode

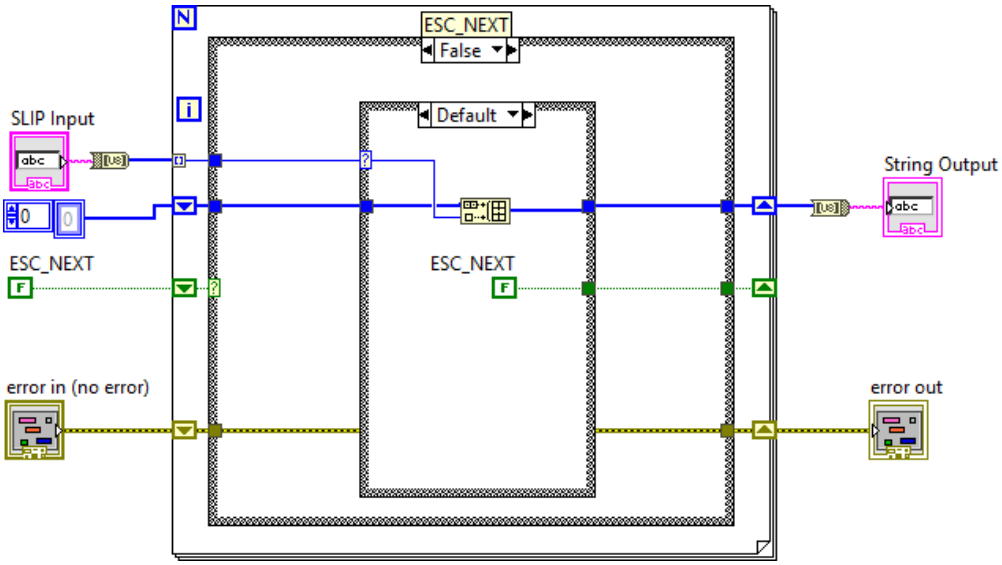

Figura 7 Caso donde no tenemos un elemento ESC y no recibimos ningún carácter especial

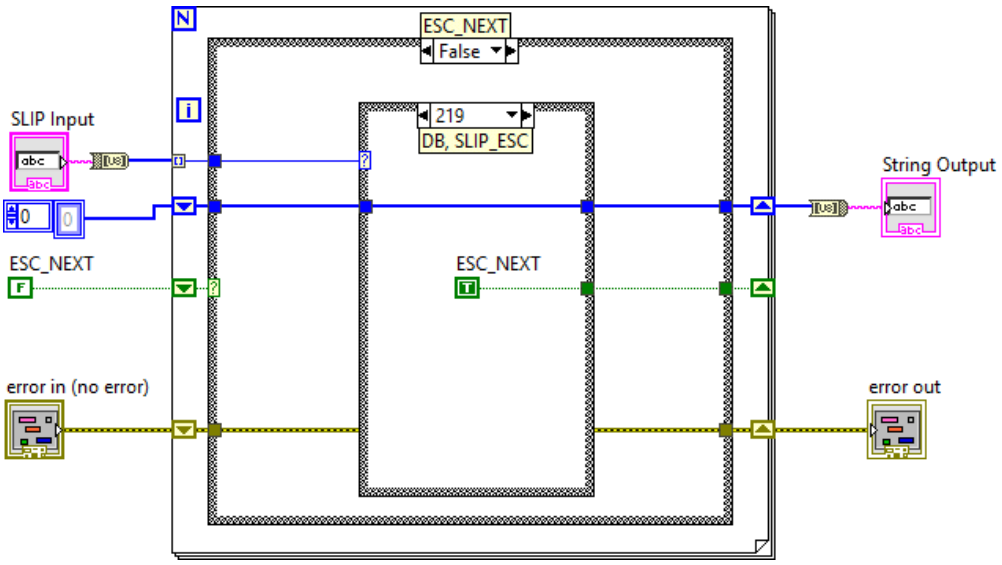

Figura 8 Caso donde no tenemos un elemento ESC y recibimos ESC

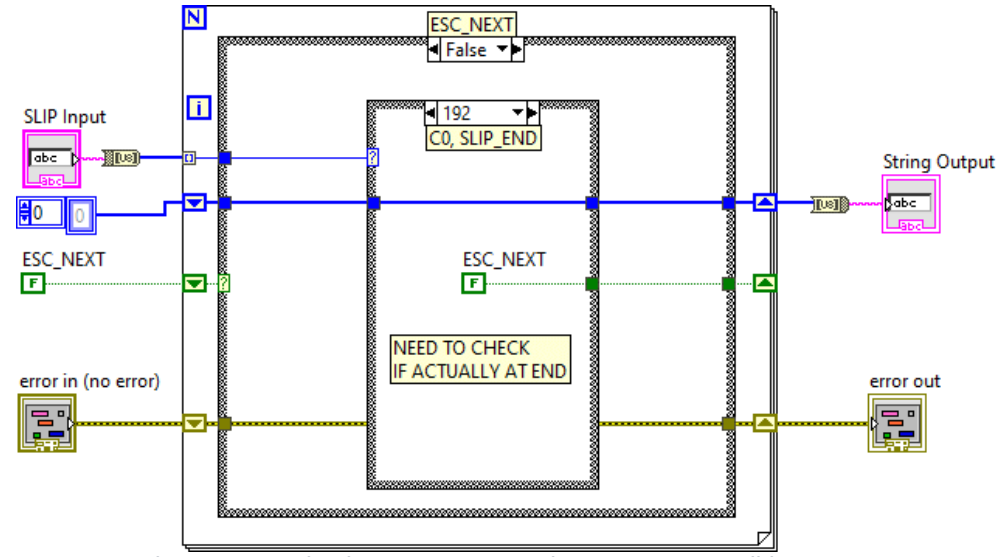

Figura 9 Caso donde no tenemos un elemento ESC y recibimos END

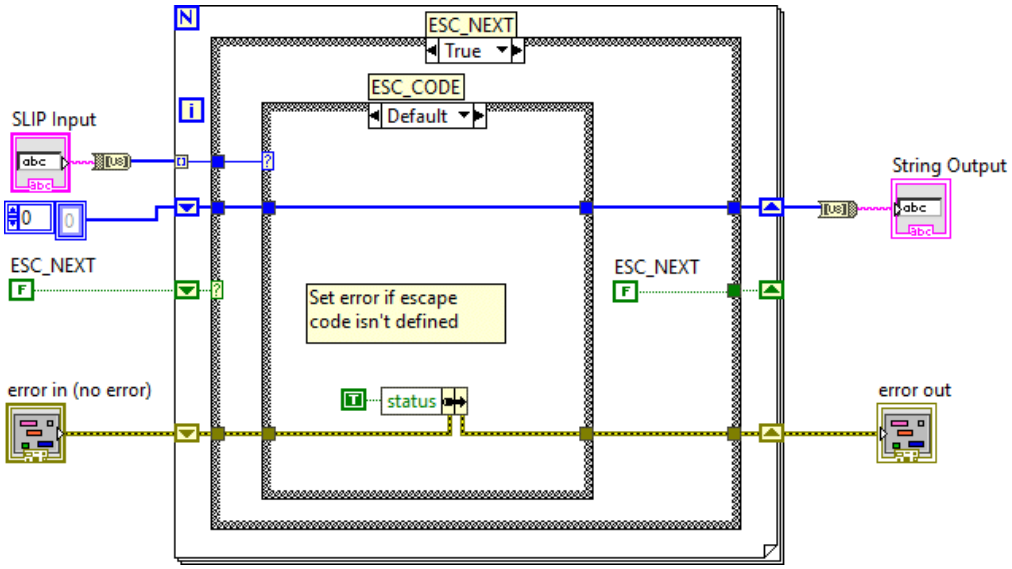

Figura 10 Caso donde tenemos un elemento ESC y no recibimos ningún carácter especial

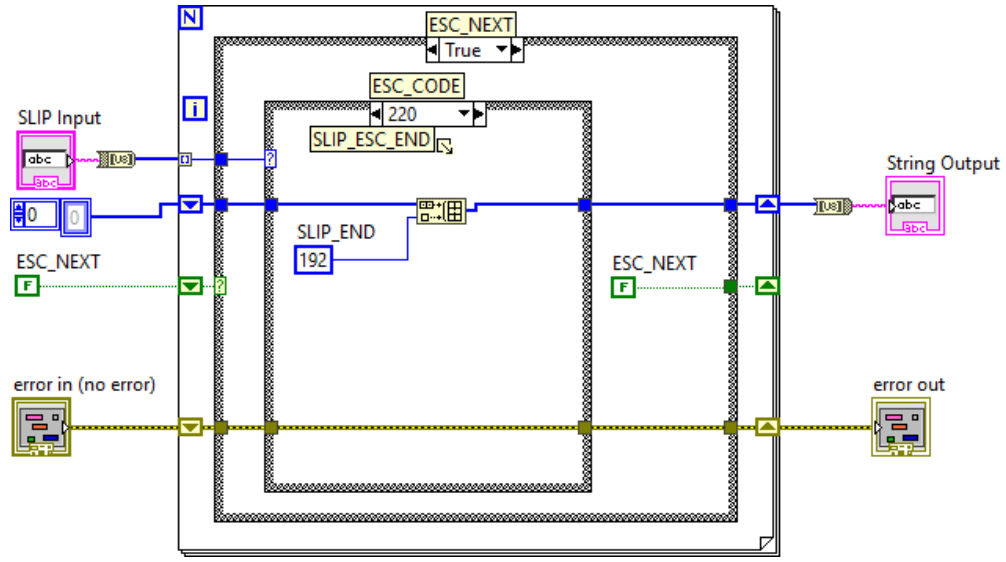

Figura 11 Caso donde tenemos un elemento ESC recibimos ESC_END

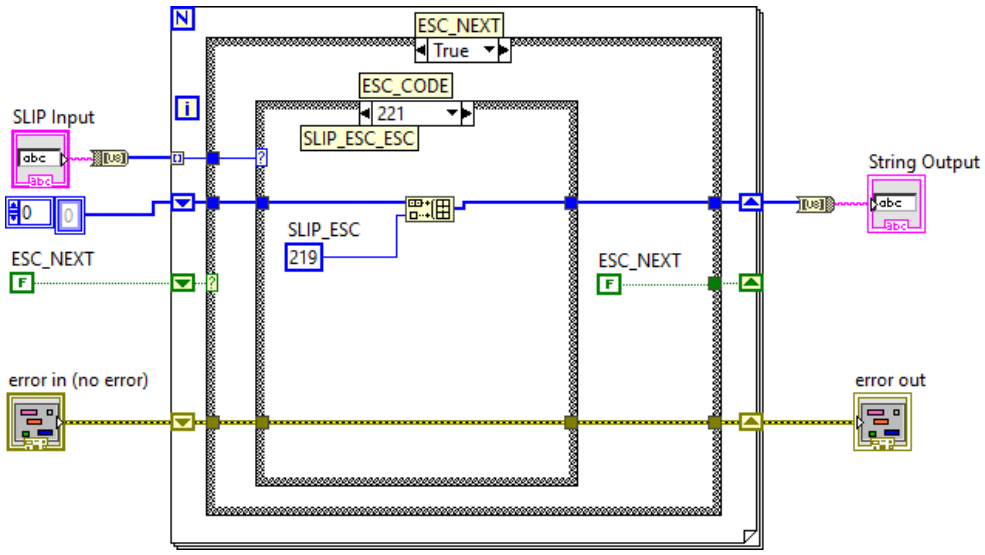

Figura 12 Caso donde tenemos un elemento ESC recibimos ESC_ESC

Al final, con todo el "algoritmo" de arriba implementado mediante máquinas de estados, una dentro de otra. Lo que se persigue es la decodificación. Tal que cuando tengamos se respeten las mismas reglas definidas en la codificación solo que a la inversa.

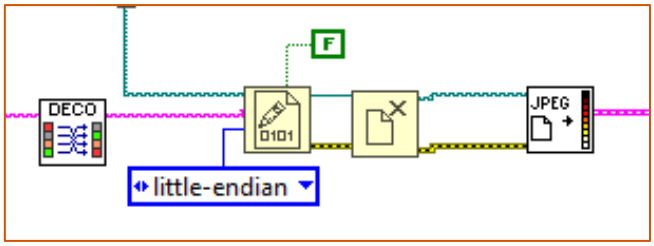

Figura 13 Escritura de bytes decodificados en un archivo binario y la generación del JPEG

Obtenida la decodificación, se procedería a convertir la imagen JPEG en formato ***imaq***. Ello permitiría trabajar con toolKit más potente (***Visión and Motion***). Una vez hecha la conversión mediante las funciones que se mostrarían a continuación, procedemos a la aplicación final.

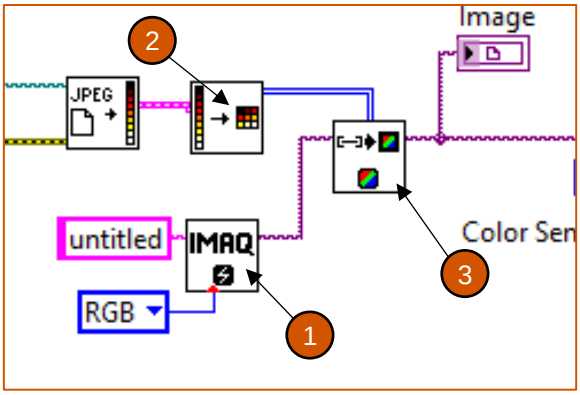

Figura 14 Conversión de JPEG a formato IMAQ

Se ha hecho uso del ***IMAQ Create*** (1) que reserva un buffer en memoria y del ***Unflatten Pixmap*** (2) que convierte el mapa de pixeles a un array 2D, a la que más tarde con el ***IMAQ ArrayToColorImage*** (3) se dibuja en el display.

Detección de color: Aplicación

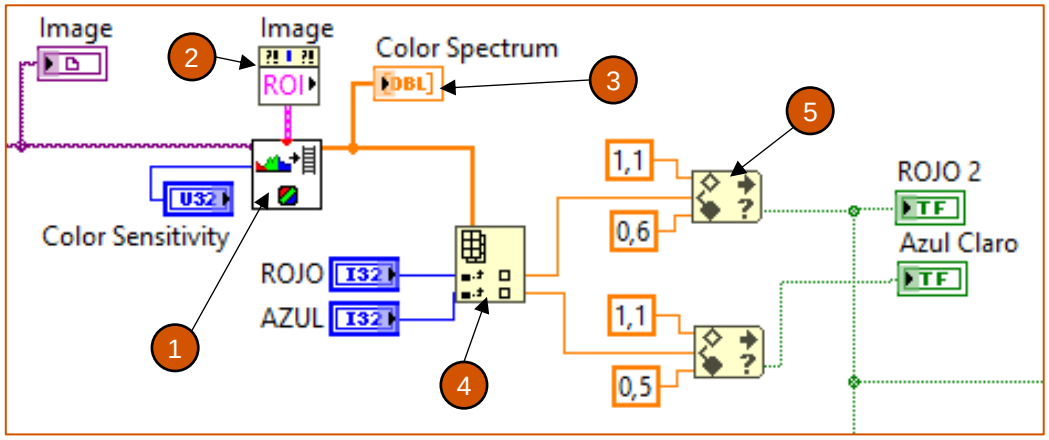

Figura 15 Detección de color

Para la detección de color se ha hecho uso del módulo ***IMAQ ColorLearn VI*** (1). Esta función del toolKit de Visión and Motion, permite conocer el color de una región de interés dada ***ROI*** (2) sobre una IMAQ. Se debe definir la sensibilidad que podría tomar valores desde ***LOW***, ***MEDIUM*** y hasta ***HIGH***. Dicha sensibilidad dividiría el circulo de colores en más sectores generando más sensibilidad a la variación de las longitudes onda detectables. Al final devuelve un array (3) con tanto elementos como se ha configurado la sensibilidad. Así en la siguiente función ***index Array*** (4), bastaría con conocer el índice interés que representa el color que se quiere detectar y recoger el valor de salida que oscilaría entre 0 (color no detectado) y 1 (máxima saturación o nivel). Ese valor se introduciría en la siguiente función ***In Range and Coerce*** (5) la cual vigila si el valor devuelto se encuentre en el rango de valores configurado y si es así devuelve un true. Así de ser el valor devuelto true, el led indicador se encendería.

El siguiente paso es implementar el contador con antirebote. Para ello se ha realizado el siguiente algoritmo.

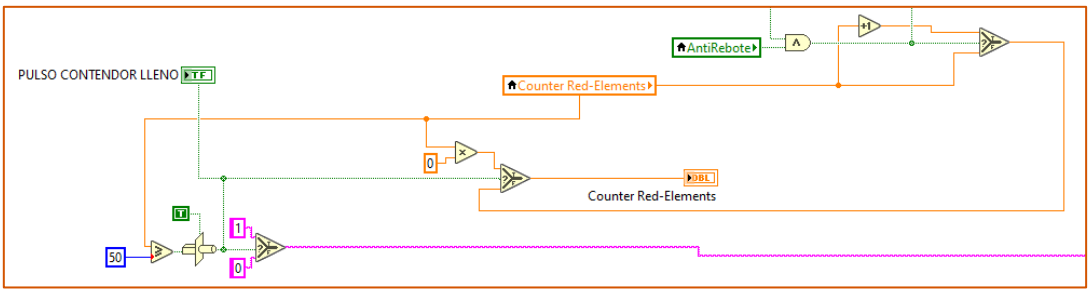

Figura 16 Contador incremental

Una vez la función ***In Range and Coerce*** (5) produce un true, se debería incrementar el contador en una unidad. No obstante, el contador se incrementaría más de una vez. Para solucionar dicho problema se ha programado un “detector de flanco de subida” como sigue.

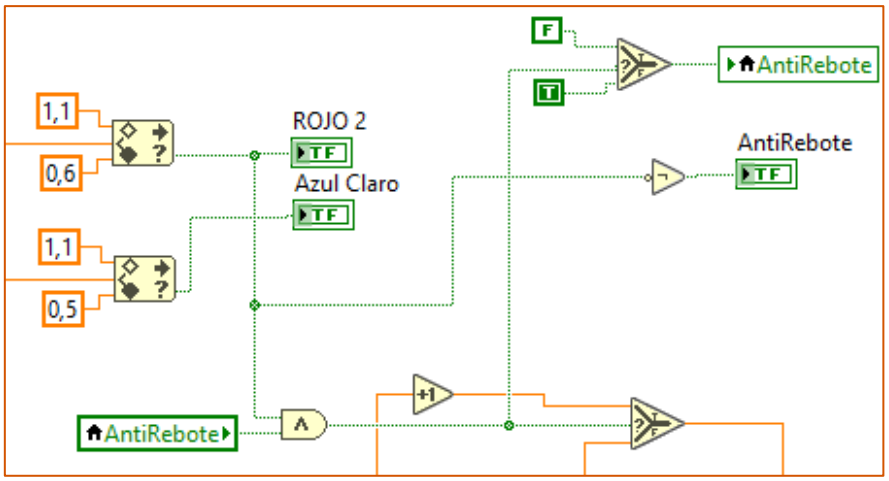

Figura 17 Condición de antirebote

Creamos un variable local a la cual llamamos ***AntiRebote*** y la inicializamos al valor negado del indicador de Led. Más tarde se va actualizando el valor de la misma mediante un selector que la pone a false cuando se ha detectado el valor y viceversa. Así, ***AntiRebote*** se pone a false tras cada detección de color. Por lo tanto, con la condición ***AND*** sólo se permite incrementar la variable una sola vez que es cuando se ha detectado el color y el ***AntiRebote*** se invierte a true, eso hace que la ***AND*** arroje un true que permite incrementar, pero al mismo tiempo llega al selector que pone a false ***AntiRebote*** impidiendo un segundo aumento consecutivo.

Finalmente, el reseteo del contador y envío de la señal al ESP32-CAM, se realiza mediante el siguiente algoritmo.

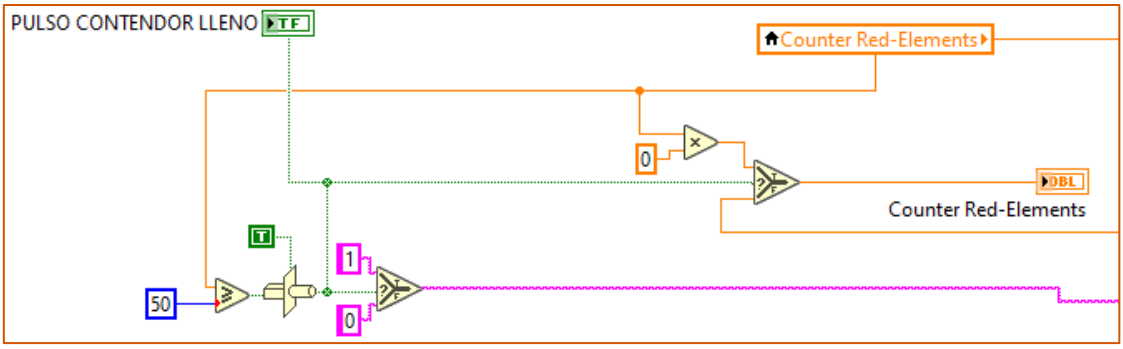

Figura 18 Reseteo y envío de señal al ESP32-CAM

Se puede observar que se ha implementado un comparador que vigila si la variable ***Counter Red-Elements*** alcanza el valor de 50, y una vez alcanzado se envía un char "1" al ESP32-CAM y se multiplica por cero el valor actual de la variable citada.

Finalmente, se implementó la función que controla el LED interno de la ESP32-CAM.

```
void ledControl(){
  char order;
  udp.parsePacket();  //Comprobar si hay paquetes udp disponibles
  udp.read(&order, sizeof(order)); //Guardar el paquete en order
  int ord= String(order).toInt(); //Pasar order("1" o "0") a int:
  digitalWrite(LED, ord); //Actuar en consecuencia con ord: 1 o 0
  delay(10); //hacer un delay de 10 ms para evitar rebotes
}
```

Resultados

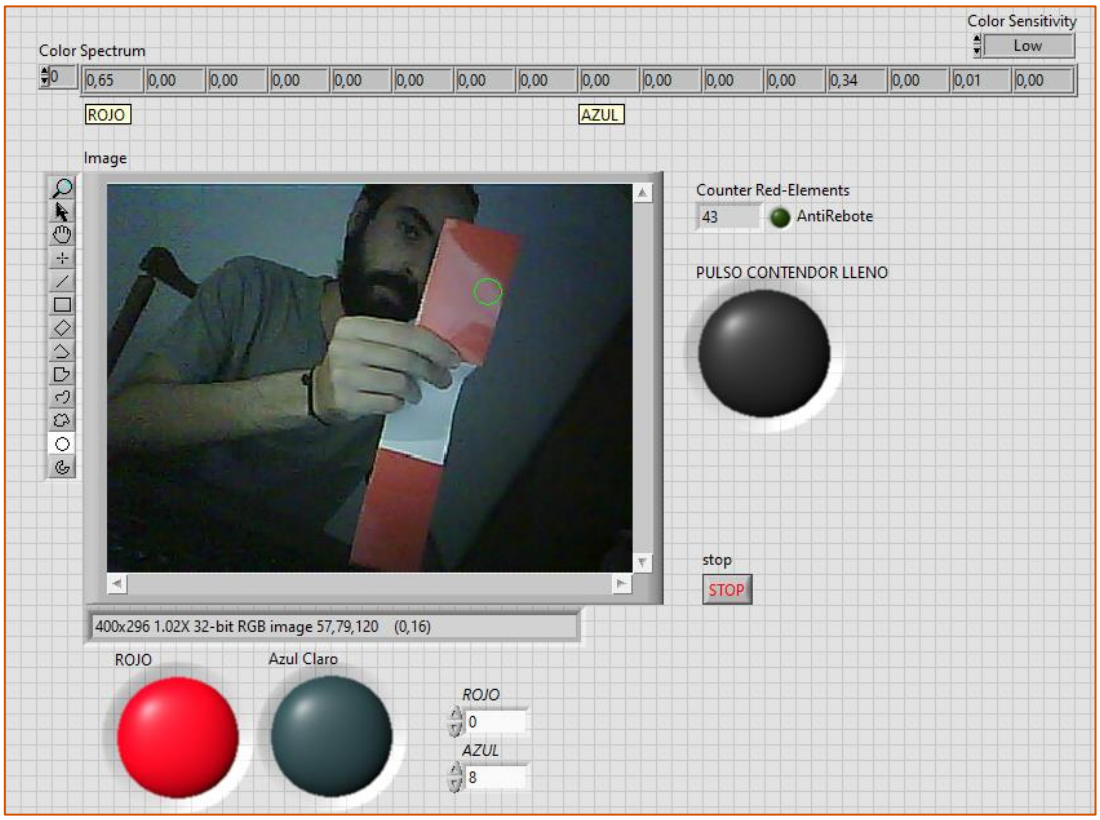

Figura 19 Panel frontal de control LabVIEW

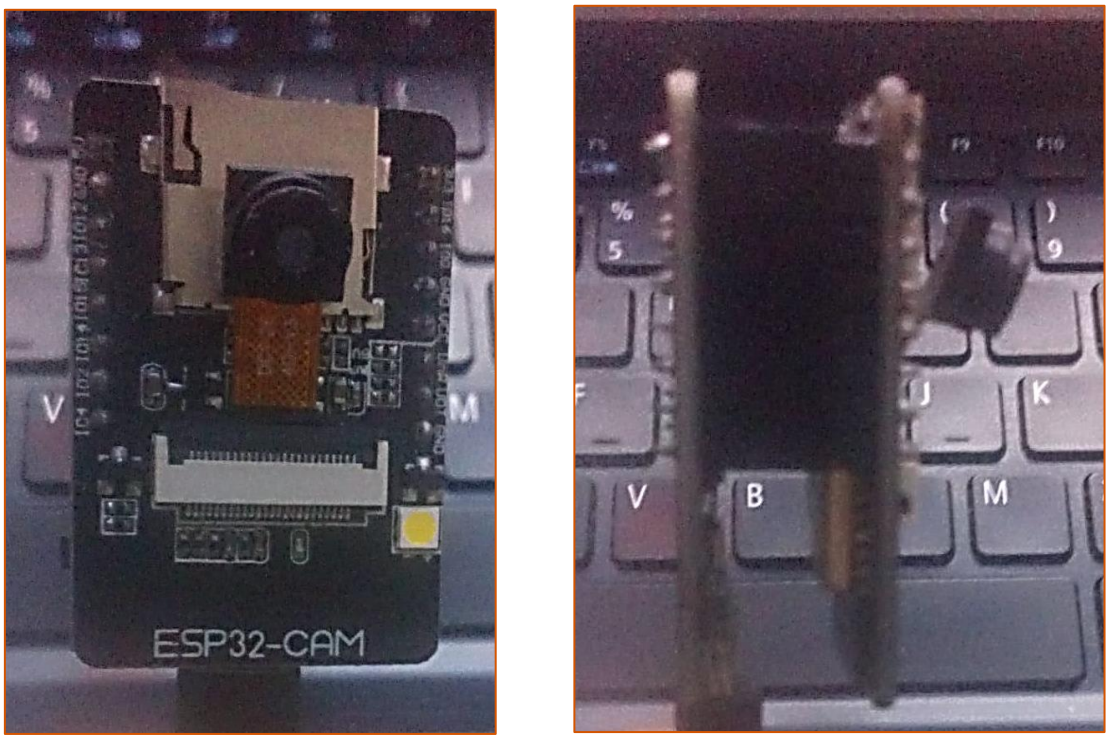

Figura 20 Módulo ESP32-CAM empleado

Conclusiones

En esta práctica se ha aprendido y se ha familiarizado con el entorno de LabVIEW, así mismo con el módulo ESP32-CAM.

En cuanto a las dificultades, uno de los problemas más grandes fue disponer de pobres recursos en el módulo (poca RAM), se hacía difícil manejar la cantidad de datos obtenidas por la captura de imágenes. Esto último, llevó a configurar la estructura que configuraba las imágenes con las mínimas prestaciones, en cuanto a calidad de compresión y tamaño. Así se tuvo que reducir el número de imágenes tomadas para el buffer a uno.

Por otra parte, el troceo que se hacía de la imagen antes de mandarla, complicó mucho la solución. Se intentó resolverlo recurriendo a la SDK (software devlopment kit) del fabricante. No obstante, resultaba complicado familiarizarse y entender el framework como para obtener una solución. En adición el factor tiempo como también las condiciones en las que se ha desarrollado el proyecto no fueron las más óptimas.

Finalmente, se recomienda realizar un análisis riguroso de las necesidades de la aplicación antes de adquirir el hardware necesario.

Bibliografía

[1] NATIONAL INSTRUMENTS CORP., «www.ni.com,» NATIONAL INSTRUMENTS CORP., 20 Junio 2022. [En línea]. Available: https://www.ni.com/es-es/shop/labview.html. [Último acceso: 20 Junio 2022].

[2] NATIONAL INSTRUMENTS CORP., «www.ni.com,» NATIONAL INSTRUMENTS CORP., 20 Junio 2022. [En línea]. Available: https://www.ni.com/es-es/innovations/white-papers/13/benefits-of-programming-graphically-in-ni-labview.html. [Último acceso: 20 Junio 2022].

[3] Ai-Thinker Co., Ltd, «docs.ai-thinker.com,» Ai-Thinker Co., Ltd, 16 Julio 2018. [En línea]. Available: https://docs.ai-thinker.com/_detail/esp32/boards/cam.65%E8%A7%84%E6%A0%BC%E4%B9%A6.jpg?id=en%3Aesp32-cam. [Último acceso: 20 Junio 2022].

[4] Ai-Thinker Co., Ltd, «docs.ai-thinker.com,» Ai-Thinker Co., Ltd, 20 Junio 2022. [En línea]. Available: https://docs.ai-thinker.com/en/esp32-cam. [Último acceso: 20 Junio 2022].

[5] D. J. W. Andrew S. Tanenbaum, Redes de computadoras, México: PEARSON EDUCACIÓN, México, 2012.