



UNIVERSIDAD TECNOLÓGICA DE PANAMA
CAMPUS LEVI LASSO



FACULTAD DE INGENIERÍA ELÉCTRICA
INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

Proyecto Final de Computadores Digitales

Tema

Brazo Robótico

Presentado Por

Fernando Guiraud 8-945-692

Eduardo Carrera 8-979-1503

Esteban Rúa, 20-14-4693

Daniel Wen 2-745-718

José Cerrud, 4-811-359

Analiz Valdés, 9-755-1964

Anays Campos, 8-970-2001

Sibeles Hansell, 1-745-104

Edwin Rodríguez, 7-709-1004

Erick Guevara, 9-758-578

Diana Méndez, 1-747-1916

Balbino Camaño, 9-754-478

Luis Ortiz, 4-778-26

Michael Saavedra, 4-765-80

Profesor
Elías Mendoza

Grupo

1EE141

Fecha de Entrega

09 de diciembre del 2022

ÍNDICE

1. Introducción
2. Objetivos del proyecto
3. Síntesis del funcionamiento
4. Programa en el microcontrolador
5. Descripción de los componentes
 - 5.1. Listado de componentes
 - 5.2. Especificaciones de cada componente
 - 5.3. Presupuesto
6. Suministro de energía
7. Esquemático completo
8. Diseño 3D
9. Descripción del código
10. Problemas presentados en el proyecto
11. Conclusiones
12. Referencias

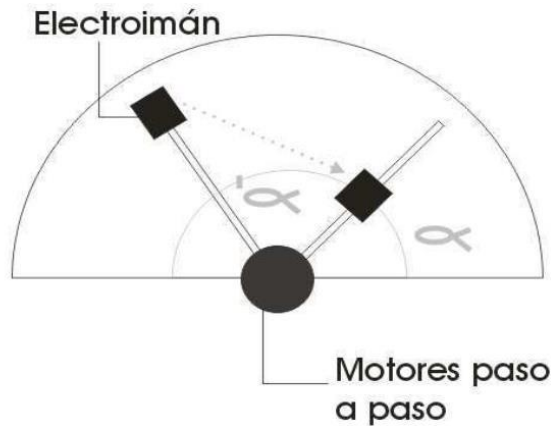
INTRODUCCIÓN

El curso de computadores digitales está destinado al control de sistemas digitales por medio de microcontroladores, en este caso hemos estudiado a fondo el control del microcontrolador ESP32, denominación de una familia de chips SoC de bajo costo y consumo de energía, con tecnología Wi-Fi y Bluetooth de modo dual integrada. El ESP32 emplea un microprocesador Tensilica Xtensa LX6 en sus variantes de simple y doble núcleo e incluye interruptores de antena, balun de radiofrecuencia, amplificador de potencia, amplificador receptor de bajo ruido, filtros, y módulos de administración de energía. El ESP32 fue creado y desarrollado por Espressif Systems y es fabricado por TSMC utilizando su proceso de 40 nm. Es un sucesor de otro SoC, el ESP8266. Este microprocesador fue programado en el entorno de Visual Studio con el uso de librerías como stdio, freertos, esp_timer y entre otras.

En este informe se detallará la elaboración del proyecto final del curso “computadores digitales”, que consiste en un brazo robótico capaz de recolectar objetos metálicos por medio de un electroimán. Este electroimán tiene una movilidad de dos grados de libertad y su posición debe ser controlada por el usuario ingresando datos al microcontrolador.

OBJETIVOS

- El proyecto consiste en mover una pieza metálica de un lugar a otro dentro de un semicírculo. Las coordenadas polares (ángulo y radio) se transmiten en forma serial desde una PC al ESP32.
-



SINTESIS DEL FUNCIONAMIENTO

1. Se establece la comunicación entre PC y microcontrolador, de manera tal que se sincronicen ambas interfaces (PC, microcontrolador). La PC espera un dato del microcontrolador. Aparece un mensaje de error en caso de no establecerse la comunicación.
2. El microcontrolador posiciona la grúa en la coordenada 0 (alfa), 0 (ro).
3. La PC pide que el usuario ingrese las coordenadas (ángulo y distancia), los transmite y el microcontrolador procesa los mismos. Luego de procesarlos el microcontrolador transmite hacia la PC los siguientes datos:
 - a) Paso alfa
 - b) Resto alfa
 - c) Paso ro
 - d) Escalón
 - e) Resto ro
 - f) Dirección
4. Una vez que el microcontrolador recibe los datos, efectúa el movimiento hacia la posición indicada por el usuario, levanta la pieza y envía un dato de fin de posición, lo que le permite al usuario poder ingresar las dos siguientes coordenadas; no es

posible ingresar las coordenadas si el proceso se está ejecutando debido a que las entradas de teclado están bloqueadas.

5. El microcontrolador le pide al usuario las coordenadas del movimiento final, procesa los datos, transmite los datos a la PC.
6. Al recibir los datos el microcontrolador efectúa el nuevo movimiento, luego deja la pieza, se vuelve a acomodar a las coordenadas 0,0 de tal manera que se reduzca los errores que se podría cometer si el proceso siguiera desde ese punto, y envía un dato a la PC que finalizó de ejecutar las instrucciones.

PROGRAMA EN EL MICROCONTROLADOR

El programa está diseñado para recibir los datos enviados por la PC y en base a ello efectuar el desplazamiento.

Los datos son los siguientes

- Paso alfa: indica la cantidad de pasos que debe dar el motor de desplazamiento angular, máximo 8 bits.
- Resto alfa: registro de 8 bits
- Paso ro: indica la cantidad de pasos que debe dar el motor de distancia, máximo 8 bits.
- Escalón: indica la cantidad de combinaciones de ambos movimientos, de alfa y ro, son necesarios para llegar a la posición deseada, máximo 8 bits.
- Resto ro: registro de 8 bits
- Dirección: es un registro de 8 bits de los cuales utilizamos los cuatro primeros bits:
 - El primer bit indica la dirección del motor de desplazamiento angular, 1 dirección hacia adelante, 0 dirección hacia atrás.
 - El segundo bit indica la dirección del motor de desplazamiento de distancia, 1 dirección hacia adelante, 0 dirección hacia atrás.
 - El tercer bit indica si se debe encender el electroimán o no, 1 encender electroimán, 0 no encender electroimán.
 - El cuarto bit indica si el brazo debe posicionarse en la posición cero o no, 1.

DESCRIPCIÓN DE LOS COMPONENTES

1. Listado de componentes

Para construir este brazo robótico se utilizaron los siguientes componentes:

ITEM	Componente	Descripción
1	Riel lineal con motor de pasos	Guía deslizante lineal 11.811 in
2	Electroimán	DC imán de solenoide electromagnético
3	Motor de pasos bipolar	Motor bipolar 2 A, 59 Ncm
4	Convertidor lógico bidireccional	Convertidor lógico bidireccional de 3.3 V a 5 V
5	Interruptor de límite de posición	
6	Microcontrolador ESP32	Microprocesador Tensilica Xtensa LX6
7	Controlador de motor de pasos TB6600	4A 9-42V Nema 17
8	Módulo de relay	

2. Especificaciones de cada componente

2.1. Riel lineal con motor de pasos



Especificación lineal de la guía de la versión del trazo
de FSL30 984.3 ft m

Especificación de la guía lineal FLS30

Dimensión y rendimiento de la guía lineal

Carrera efectiva: 11.811 in.

Cantidad de diapositivas: 1

Modelo de tornillo: Tr8 x 2.

Vida del movimiento: 10.000 horas.

Carga horizontal máxima: 6.6 libras

Carga vertical máxima: 2.2 libras

Velocidad máxima horizontal total: 2.756 in/s

Velocidad máxima vertical completa: 1.969 in/s

Velocidad máxima sin carga: 3.150 in/s.

Ruido: 65 dB (dentro de 3.3 ft)

Sobre este artículo

- Carga horizontal máxima de 8.8 lbs mientras que carga vertical máxima de 2.2 lbs
- Precisión de posición repetida de 0.002 in
- Trazo efectivo: 11.811 in, tiene múltiples opciones de 1.969 in a 11.811 in

- Tornillo Tr8 x 2 fabricado como modelo de tornillo predeterminado, tiene Tr8 x 2, Tr8 x 4, Tr8 x 12 modelo de tornillo para elegir, ponte en contacto con nosotros cuando necesites cambiar el modelo de tornillo
- El motor paso a paso Nema 14 duradero tiene 10000 horas de vida útil, tiene otro motor para elegir, ponte en contacto con nosotros cuando necesites cambiar el motor.

2.2. Electroimán

Electroimán de elevación eléctrico de 12 V CC 180 N electroimán de elevación

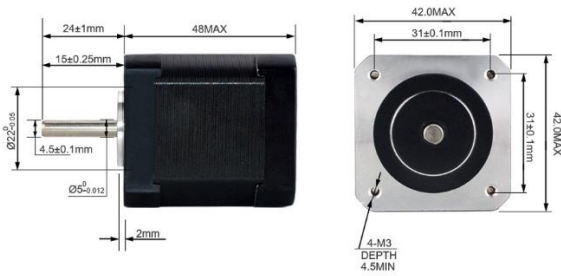


Sobre este artículo:

- Tamaño: voltaje de entrada: CC 12 V; succión: 39.7 lbs/180 N; tamaño total: 1.339 x 0.709 in (diámetro x altura). Modelo: ZYE1-P3418
- Características: el electroimán de sujeción es simple y fiable en diseño de estructura, fácil de fijar y conectar a la carga, y tiene baja fricción para garantizar una alta eficiencia, aumento de temperatura estable, prolongar la vida útil del producto y garantizar un buen rendimiento
- Ventaja: el imán de elevación eléctrico tiene una superficie plana y lisa, excelente mano de obra, buena conductividad eléctrica y aislamiento, tamaño pequeño y gran fuerza de absorción, es un producto potente, compacto, de bajo consumo y fiable
- Aplicación: el electroimán de sujeción es ampliamente utilizado en la automatización, como líneas de montaje, máquina de clasificación, brazo mecánico, instalación experimental, máquinas expendedoras, etc. Adecuado para equipos de transporte, instalaciones de oficina, electrodomésticos, cerraduras automáticas de puertas, etc

Nota: la fuerza se reducirá debido al trabajo prolongado, menor voltaje y objeto aspirado inadecuado, etc., se recomienda que la fuerza real sea inferior al 80% de la fuerza aspirada en la placa de nombre; el electroimán en condiciones de trabajo produce una cierta cantidad de calor, electricidad con más frecuencia la temperatura más alta, lo que es un fenómeno normal

2.3. Motor de pasos bipolar



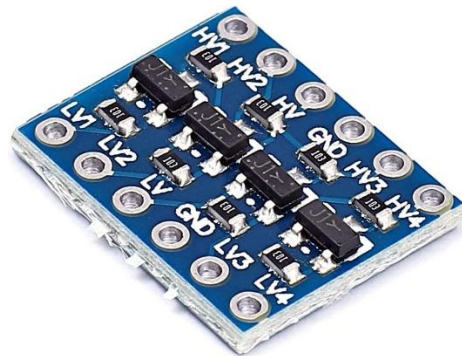
Motor Type: Bipolar Stepper
Step Angle: 1.8 deg
Holding Torque: 59Ncm(84oz.in)
Rated Current/phase: 2.0A
Phase Resistance: 1.4ohms
Inductance: 3.0mH \pm 20%(1KHz)

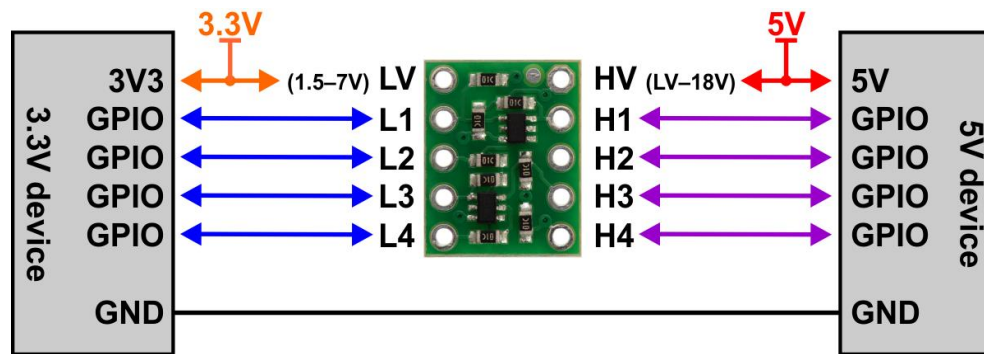


Sobre este artículo

- Motor de impresora 3D con torque alto.
- 83.6 oz pulgadas de torque de sostenimiento.
- Nema 17 bipolar con 4 cables de 1.65 pulgadas x 1.65 pulgadas x 1.85 in
- Construye con cable de 1.1 yd y conector de paso de 0.1 pulgadas.
- Corriente nominal de 2.0 A y resistencia de 1.4 ohm

2.4. Convertidor lógico bidireccional





Sobre este artículo

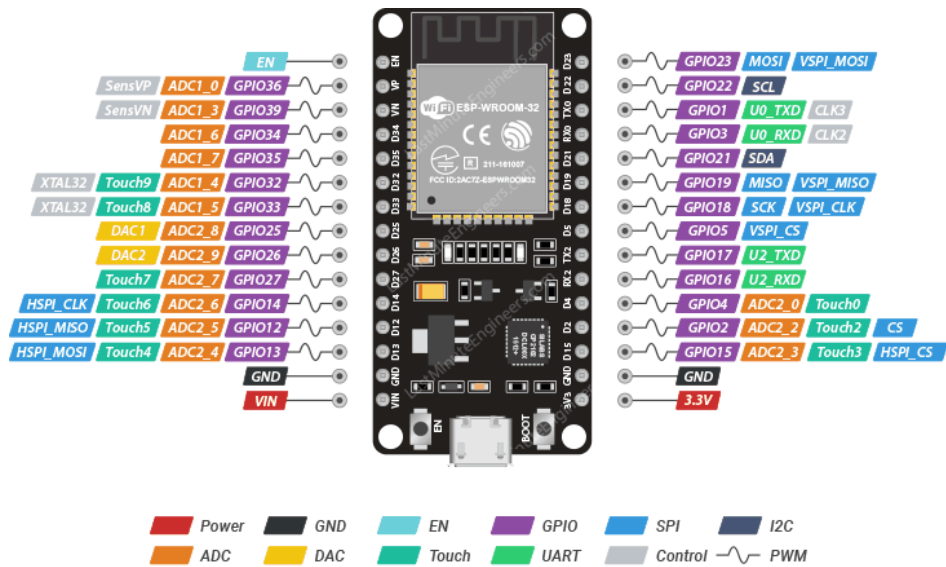
- El convertidor de nivel lógico bidireccional es un dispositivo pequeño, que puede bajar de forma segura señales de 5 V a 3,3 V y subir de 3,3 V a 5 V al mismo tiempo.
- Este convertidor de nivel funciona con dispositivos de 2.8 V y 1.8 V, puede transferir bidireccionalmente con 4 canales entre alto voltaje lógico y bajo voltaje lógico.
- Cada convertidor de nivel lógico tiene la capacidad de convertir 4 pines en el lado alto a 4 pines en el lado bajo con dos entradas y dos salidas proporcionadas para cada lado.

2.5. Interruptor de limite



Estos interruptores cumplen la función de detectar las posiciones extremas del recorrido del riel lineal y el motor angular. Son utilizados para la creación de una subrutina de calibración que se ejecuta al inicio y final de cada recolección.

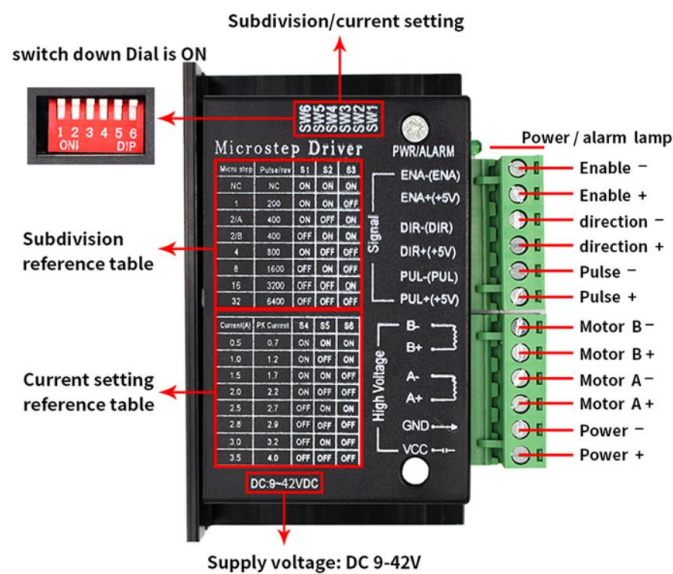
2.6. Microcontrolador ESP32



Sobre este artículo

- WiFi de modo dual de 2,4 GHz + placa de desarrollo Bluetooth
- Consumo de energía ultra bajo, funciona perfectamente con el Arduino IDE
- Compatible con protocolo LWIP, Freertos
- Compatible con tres modos: AP, STA y AP+STA
- ESP32 es seguro, fiable y escalable a una variedad de aplicaciones

2.7. Controlador de motor de pasos TB6600



Este controlador es una versión mejorada del TB6600

Controlador de motor paso a paso bipolar tipo chopper PWM IC

- Fuente de alimentación de 9 V-42 V CC; Unidad de flujo de fase constante bipolar de puente H.
- El máximo de 4,0 A de los ocho tipos de corriente de salida.
- Las 32 subdivisiones más grandes de los 6 modelos son opcionales.
- Señal de entrada de aislamiento fotoeléctrico de alta velocidad.
- Interfaz de pulso único estándar de ánodo común. Fuera de línea para mantener la función.
- Protección de temperatura integrada y protección contra sobre corriente.
- Para proporcionar la función de bloqueo de corriente semiautomática de ahorro de energía.
- El recinto semicerrado se puede adaptar a un entorno más estricto.
- Solo la señal de pulso del reloj puede ser impulsada por un motor paso a paso bipolar de dos fases para lograr un trabajo de baja vibración y alta eficiencia.
- Puede controlar la rotación hacia adelante y hacia atrás del motor paso a paso de 2 fases, con modo de excitación de fase 1-2, W1-2, 2W1-2, 4W1-2.

2.8. Relay module



La función de este relay es encender el electroimán por medio de una señal digital que es enviada por el controlador para determinar cuándo debe agarrar o soltar la pieza metálica que se encuentre levantando.

PRESUPUESTO

Compra de Amazon de los componentes

Componente	Cantidad	Precio	ITBMS	Costo
Guía deslizante lineal 11.811 in	1	\$ 118.00	\$ 8.26	\$ 126.26
Motor bipolar 2 A, 59 Ncm	1	\$ 13.99	\$ 0.98	\$ 14.97
Controlador de motor paso a paso TB6600 4A 9-42V	2	\$ 9.98	\$ 0.70	\$ 21.36
DC imán de solenoide electromagnético	1	\$ 12.99	\$ 0.91	\$ 13.90
Convertidor de nivel lógico bidireccional de 3.3 V a 5 V	1	\$ 5.99	\$ 0.42	\$ 6.41
			Total	\$ 182.90

Costos de envío	Tracking	Peso (lb)	Costo
Paquete #1	TBA303527124673	4	\$ 10.00
Paquete #2	TBA303451041633	1	\$ 2.50
Paquete #3	TBA303540170807	1	\$ 2.50
		Total	\$ 15.00

Resumen de total de costos del proyecto:

Componente	Costo
Riel lineal con motor de pasos	\$ 126.26
Motor de pasos bipolar	\$ 14.97
Controlador de motod paso a paso TB6600	\$ 21.36
Electroiman solenoide de 12V	\$ 13.90
Relay	\$ 5.00
Convertidores elevadores bidireccionales	\$ 6.41
Microcontrolador ESP32	\$ 10.99
Interruptores de limite de posición	\$ 3.00
Filamento de impresión 3D	\$ 44.66
Pegamento epoxico	\$ 2.54
Soportes para mesa	\$ 2.99
Tabla de plywood	\$ 8.00
Envios de Amazon	\$ 15.00
Total	\$ 275.08

SUMINISTRO DE ENERGÍA

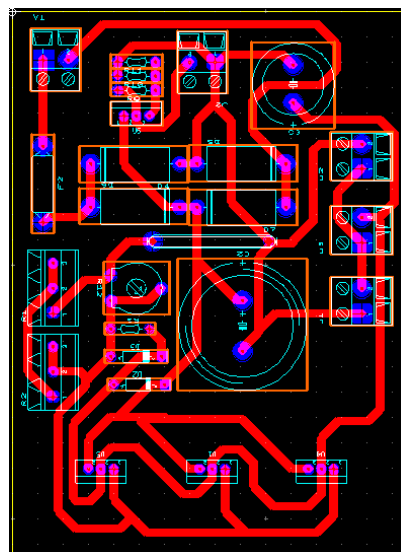
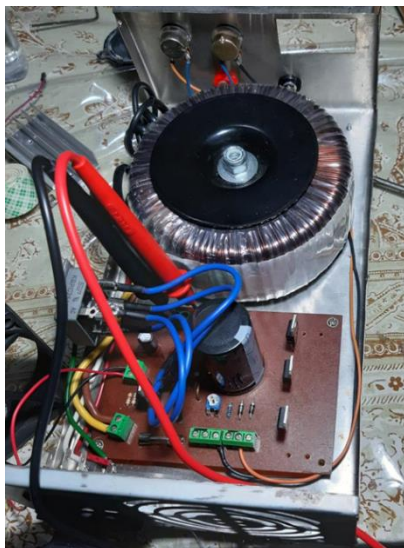
El suministro de energía de este brazo robótico principalmente está determinado por el consumo de los motores de pasos, ya que son los de mayor consumo. La corriente de operación del motor de pasos del riel lineal es de 800 miliamperios y del motor de control angular es de 2 amperios. El electroimán tiene un consumo de 500 miliamperios. Todos estos componentes funcionan a 12 voltios, por lo que la fuente de alimentación del proyecto debe corresponder a esta tensión. Para resumir las potencias totales de este proyecto desarrollamos la siguiente tabla:

Dispositivo	Voltaje (V)	Corriente (A)	Potencia (W)
Motor de riel	12	0.8	9.6
Motor angular	12	2	24
Electroimán	12	0.5	6
ESP32	5	0.05	0.25
	Total	3.35	39.85

La corriente máxima de operación es de 3.35 Amperios, pero asumiendo un margen de seguridad en el caso de que los motores se vean forzados por cualquier efecto de fricción o torsión del eje y no llevar la fuente de alimentación a sus condiciones de operación máximas.

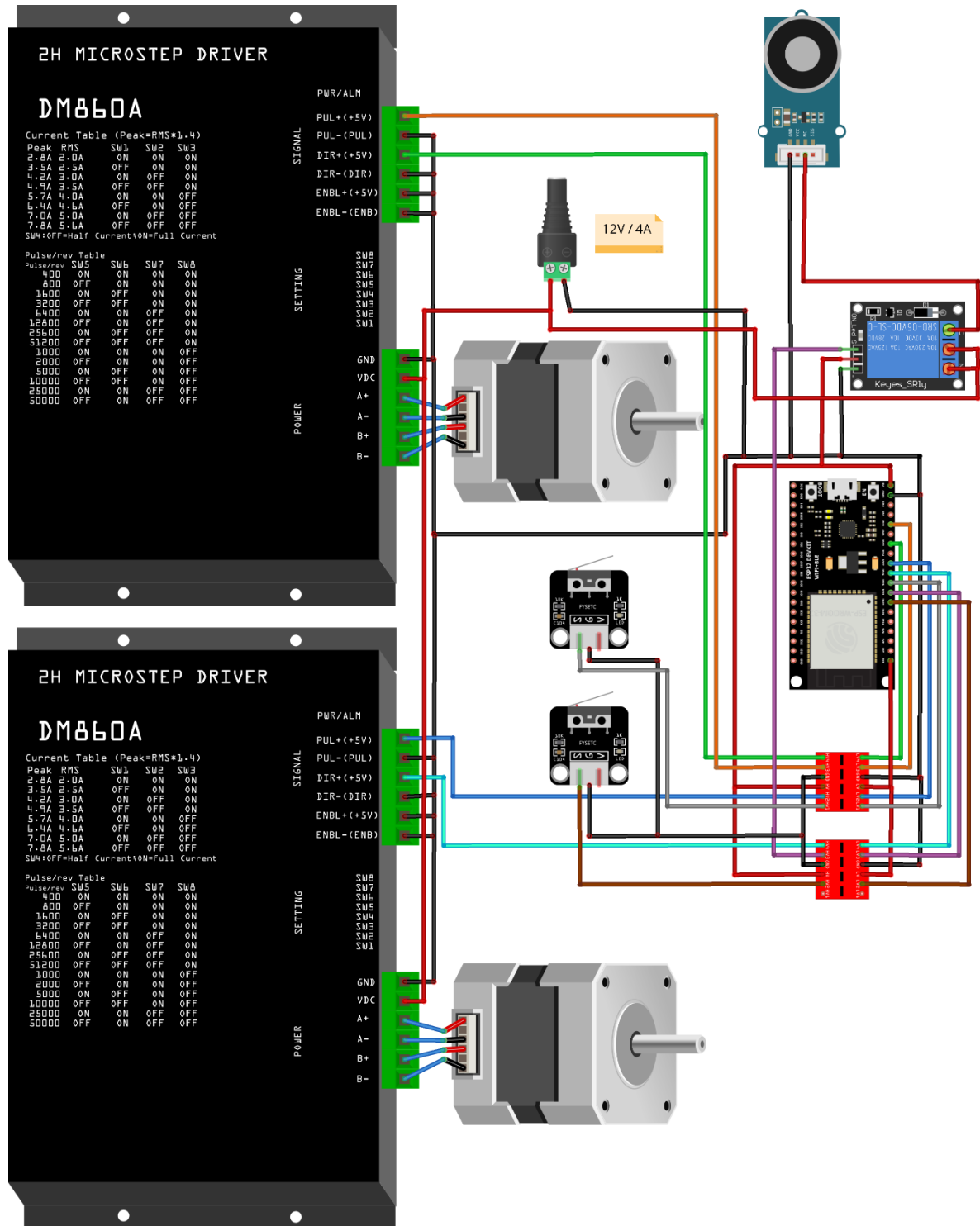
La alimentación del microcontrolador ESP32 es suministrada por la computadora a través de la conexión en el puerto USB destinado a la comunicación serial de los datos ingresados por el usuario.

Para el suministro de energía de los motores a una tensión de 12 voltios, se utilizó una fuente de alimentación de 10 amperios regulable de 0 a 30 voltios, la cual nos permite tener un amplio margen de potencia para el control de los motores.



ESQUEMÁTICO COMPLETO

En el siguiente diagrama podemos ver todas las conexiones de cada uno de los componentes al microcontrolador ESP32 y su alimentación.

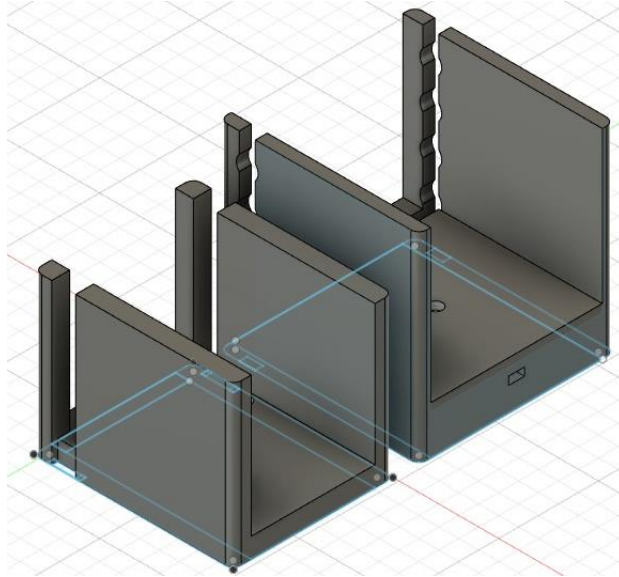


fritzing

DISEÑO 3D

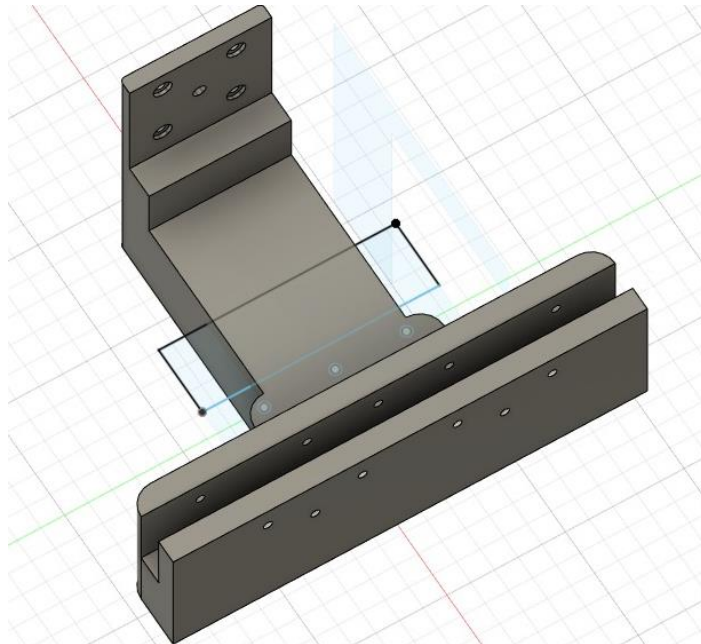
Para la elaboración de este proyecto, se requirió ensamblar múltiples componentes, por lo que se modeló en 3D cada una de las piezas que serán detalladas a continuación:

1. Soporte del electroimán en el riel



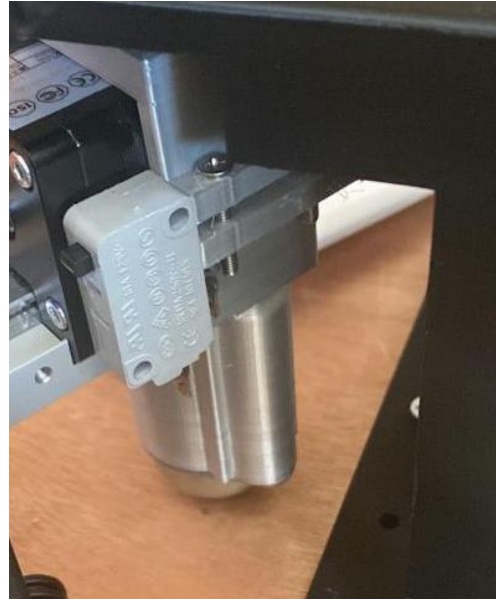
La primera pieza diseñada fue la conexión entre el electroimán y el riel. Fue diseñada como dos C ajustables con tornillos de forma que se pueda escoger un tamaño de muestra a recolectar dependiendo de su altura.

2. Soporte del motor de pasos de movimiento angular



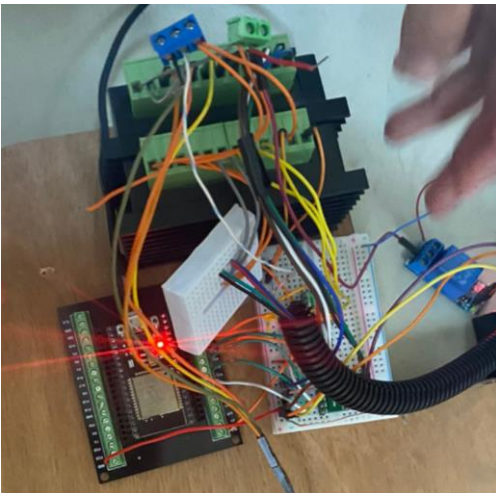
Ese es el soporte que mayor tensión debe soportar, ya que el motor que ejerce el toque para producir la rotación del brazo robótico se encuentra sobre esta pieza. Además, en la parte inferior del motor, se cuelga el riel por lo que todo el peso recae en el ángulo. El soporte se ancla a la base de plywood inferior y se atornilla en los 4 agujeros de abajo para evitar vibraciones o cualquier otro movimiento.

3. Ruedas locas deslizantes del riel



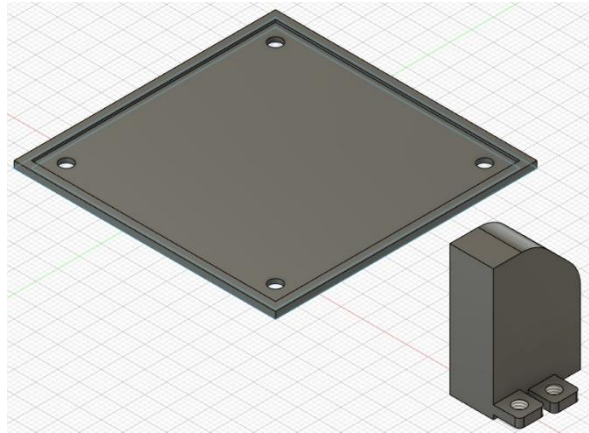
Son el punto de pivote y deslizamiento del riel sobre la superficie, cilindros impresos en 3D con ruedas con un balín en su interior que permite la movilidad en múltiples direcciones.

4. Caja para protección del circuito eléctrico



Con el fin de acomodar los cables del circuito y evitar que alguno de estos se desconectara en el funcionamiento, se diseñó un cobertor para tapar todos los circuitos incluyendo el microcontrolador ESP32, dejando únicamente los disipadores de los controladores afuera.

5. Soporte para el interruptor de posición inicial del motor angular y inicio



Se diseñó una pequeña pieza que se encuentra en el extremo del recorrido, donde se conecta atornillado un interruptor limitante que permite determinar cuando los motores se encuentran en sus posiciones iniciales.

6. Ensamblaje final de todos los soportes



Por último, se colocaron todas las piezas impresas en 3D a la medida y se ajustaron todas las imperfecciones en el acabado de las impresiones.

DESCRIPCIÓN DEL CÓDIGO

Se elaboro el siguiente código en Visual Studio Code:

```
// librerias c
#include <stdio.h>
#include <string.h>
// librerias esp32
#include "driver/gpio.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_timer.h"
#include "driver/uart.h"
#include "esp_heap_caps.h"

// pin definitions
const int lims_pin[2] = {32, 25};
#define lim_pin_ro 32
#define lim_pin_alpha 25
const int ena_pin[2] = {-1, -1};
const int dir_pin[2] = {26, 12};
const int pul_pin[2] = {27, 13};
const int iman_pin = 33;

// motor id
#define ALPHA 1
#define RO 0
// direcciones
#define CW 1
#define CCW 0

// Conversion cm a steps
int cm_to_steps(int cm)
{
    return cm * 100;
}

// Conversion grados a steps
int degree_to_steps(int alpha)
{
    return (alpha * (800 / 360)) + 37;
}

// pin bit mask
#define gpio_input_mask (1ULL << lims_pin[ALPHA]) | (1ULL << lims_pin[RO])
```

```
#define gpio_output_mask (1ULL << dir_pin[ALPHA]) | (1ULL << pul_pin[ALPHA])  
| (1ULL << dir_pin[R0]) | (1ULL << pul_pin[R0]) | (1ULL << iman_pin)  
  
/* inicializacion del hardware  
* modo output  
    - no interrupciones  
    - no pulldown  
    - no pullup  
* modo input  
    - no interrupciones  
    - pulldown  
    - no pullup  
*/  
void init_hw(void)  
{  
    gpio_config_t io_config;  
  
    io_config.pin_bit_mask = gpio_output_mask;  
    io_config.mode = GPIO_MODE_OUTPUT;  
    io_config.intr_type = GPIO_INTR_DISABLE;  
    io_config.pull_down_en = GPIO_PULLDOWN_DISABLE;  
    io_config.pull_up_en = GPIO_PULLUP_DISABLE;  
    gpio_config(&io_config);  
  
    io_config.pin_bit_mask = gpio_input_mask;  
    io_config.mode = GPIO_MODE_INPUT;  
    io_config.intr_type = GPIO_INTR_DISABLE;  
    io_config.pull_down_en = GPIO_PULLDOWN_ENABLE;  
    io_config.pull_up_en = GPIO_PULLUP_DISABLE;  
    gpio_config(&io_config);  
  
    for (int i = 0; i < 2; i++)  
    {  
        gpio_set_level(dir_pin[i], 0);  
        gpio_set_level(pul_pin[i], 0);  
    }  
    gpio_set_level(iman_pin, 1);  
}  
  
const uart_port_t uart_num = UART_NUM_0;  
#define uart_buffer_size (1024 * 2)  
QueueHandle_t uart_queue;  
void init_uart(void)  
{
```

```
uart_config_t uart_config = {
    .baud_rate = 115200,
    .data_bits = UART_DATA_8_BITS,
    .parity = UART_PARITY_DISABLE,
    .stop_bits = UART_STOP_BITS_1,
    .flow_ctrl = UART_HW_FLOWCTRL_DISABLE};

uart_param_config(uart_num, &uart_config);
uart_set_pin(uart_num, -1, -1, -1, -1);
uart_driver_install(uart_num, uart_buffer_size, uart_buffer_size, 10,
&uart_queue, 0);
}

#define period_ro 1
#define period_alpha 3

int *i;
/* funcion de operacion del stepper
 * steps = cantidad de pulsos del reloj = cantidad de pasos del motor,
 * dir = direccion de giro
 * motor = id del motor
 */
void steps(int step, int dir, int motor)
{
    // asignar memoria
    i = heap_caps_calloc(1, 2, MALLOC_CAP_8BIT);

    // motor id
    if (motor == R0)
        step = cm_to_steps(step);
    else if (motor == ALPHA)
        step = degree_to_steps(step);
    else
        step = 0;

    // determinar direccion
    gpio_set_level(dir_pin[motor], dir);

    // tren de pulsos
    if (motor == ALPHA)
    {
        for (i[0] = 0; i[0] < step; i[0]++)
        {
            gpio_set_level(pul_pin[motor], 1);
        }
    }
}
```

```
        vTaskDelay(period_alpha / portTICK_PERIOD_MS);
        gpio_set_level(pul_pin[motor], 0);
        vTaskDelay(period_alpha / portTICK_PERIOD_MS);
    }
    // frenado
    gpio_set_level(dir_pin[motor], !dir);
    for (i[0] = 0; i[0] < 30; i[0]++)
    {
        gpio_set_level(pul_pin[motor], 1);
        vTaskDelay(period_alpha / portTICK_PERIOD_MS);
        gpio_set_level(pul_pin[motor], 0);
        vTaskDelay(period_alpha / portTICK_PERIOD_MS);
    }
}
else
{
    for (i[0] = 0; i[0] < step; i[0]++)
    {
        gpio_set_level(pul_pin[motor], 1);
        vTaskDelay(period_ro / portTICK_PERIOD_MS);
        gpio_set_level(pul_pin[motor], 0);
        vTaskDelay(period_ro / portTICK_PERIOD_MS);
    }
}
}

int *limit_level_1; // limit switch alpha
int *limit_level_2; // limit switch ro
int prev_r = 0;     // valor R0 previo
int prev_a = 0;     // valor ALPHA previo
/* funcion de calibracion
 * alpha hasta tocar limit_switch
 * ro hasta tocar limit_switch
 */
void calibrate(void)
{
    // memory allocation
    limit_level_1 = heap_caps_malloc(1, 1, MALLOC_CAP_8BIT);
    limit_level_2 = heap_caps_malloc(1, 1, MALLOC_CAP_8BIT);
    i = heap_caps_malloc(1, 1, MALLOC_CAP_8BIT);

    // calibracion ro
    limit_level_2[0] = gpio_get_level(lim_pin_ro);
    gpio_set_level(dir_pin[R0], CCW);
```

```
while (limit_level_2[0] == 0)
{
    gpio_set_level(pul_pin[R0], 1);
    vTaskDelay(period_ro / portTICK_PERIOD_MS);
    gpio_set_level(pul_pin[R0], 0);
    vTaskDelay(period_ro / portTICK_PERIOD_MS);

    limit_level_2[0] = gpio_get_level(lim_pin_ro);
}

vTaskDelay(2000 / portTICK_PERIOD_MS);

// calibracion alpha
limit_level_1[0] = gpio_get_level(lim_pin_alpha);
gpio_set_level(dir_pin[ALPHA], CCW);
while (limit_level_1[0] == 1) // frecuencia de calibracion lenta
{
    gpio_set_level(pul_pin[ALPHA], 1);
    vTaskDelay(period_alpha / portTICK_PERIOD_MS);
    gpio_set_level(pul_pin[ALPHA], 0);
    vTaskDelay(period_alpha / portTICK_PERIOD_MS);

    limit_level_1[0] = gpio_get_level(lim_pin_alpha);
}
// frenado
for (i[0] = 0; i[0] < 10; i[0]++)
{
    gpio_set_level(pul_pin[ALPHA], 1);
    vTaskDelay(3 / portTICK_PERIOD_MS);
    gpio_set_level(pul_pin[ALPHA], 0);
    vTaskDelay(3 / portTICK_PERIOD_MS);
}

vTaskDelay(1000 / portTICK_PERIOD_MS);

prev_a = 0;
prev_r = 0;

// liberar memoria
heap_caps_free(limit_level_1);
heap_caps_free(limit_level_2);
heap_caps_free(i);
}
```

```
uint8_t *uart_data; // bus para recibir datos del uart
uint8_t *a_index;   // indice de letra a
uint8_t *f_index;   // indice caracter ';'
char *rc_val;       // string con el valor en R0
char *ac_val;       // string con el valor en ALPHA
int *r;             // int con valor en R0
int *a;             // int con valor en ALPHA
void get_uart_values(int length)
{
    // asignar memoria
    i = heap_caps_malloc(1, 1, MALLOC_CAP_8BIT);
    a_index = heap_caps_malloc(1, 1, MALLOC_CAP_8BIT);
    f_index = heap_caps_malloc(1, 1, MALLOC_CAP_8BIT);
    rc_val = heap_caps_malloc(3, 1, MALLOC_CAP_8BIT);
    ac_val = heap_caps_malloc(3, 1, MALLOC_CAP_8BIT);

    // obtener indices de instruccion: r###a###
    for (i[0] = 0; i[0] < length; i[0]++)
    {
        if ((char)uart_data[i[0]] == 'a')
            a_index[0] = i[0];
        if ((char)uart_data[i[0]] == ';')
            f_index[0] = i[0];
    }

    // convertir a int el valor de desplazamiento en R0
    for (i[0] = 0; i[0] < a_index[0] - 1; i[0]++)
    {
        rc_val[i[0]] = (char)uart_data[i[0] + 1];
    }
    sscanf(rc_val, "%d", &r[0]);

    // convertir a int el valor de desplazamiento en ALPHA
    for (i[0] = a_index[0]; i[0] < f_index[0]; i[0]++)
    {
        ac_val[i[0] - a_index[0]] = (char)uart_data[i[0] + 1];
    }
    sscanf(ac_val, "%d", &a[0]);

    // liberar memoria
    heap_caps_free(i);
    heap_caps_free(a_index);
    heap_caps_free(f_index);
    heap_caps_free(rc_val);
}
```

```
    heap_caps_free(ac_val);
}

int cnt = 0; // contador de fase de rutina
/* Funcion de rutina => buscar, recoger, regresar
 * hace un string-to-int para los valores obtenidos del uart
 * para la distancia en R0 y el angulo en ALPHA
 * luego ejecuta la rutina
 */
void rutina(int length)
{
    if (cnt == 0)
    {
        // buscar
        steps(r[0], CW, R0);
        vTaskDelay(1000 / portTICK_PERIOD_MS);
        steps(a[0], CW, ALPHA);
        // agarrar
        gpio_set_level(iman_pin, 0);

        cnt = 1;
        prev_a = a[0];
        prev_r = r[0];
    }
    else if (cnt == 1)
    {
        uint8_t *n_dir_ro = heap_caps_malloc(1, 1, MALLOC_CAP_8BIT);
        uint8_t *n_dir_alpha = heap_caps_malloc(1, 1, MALLOC_CAP_8BIT);
        uint8_t *n_alpha = heap_caps_malloc(1, 1, MALLOC_CAP_8BIT);
        uint8_t *n_ro = heap_caps_malloc(1, 1, MALLOC_CAP_8BIT);

        if (prev_a < a[0]) // si el nuevo alpha es mayor (movimiento
izquierda)
        {
            n_dir_alpha[0] = CW;
            n_alpha[0] = a[0] - prev_a;
        }
        else
        {
            n_dir_alpha[0] = CCW;
            n_alpha[0] = prev_a - a[0];
        }
        if (prev_r < r[0]) // si nuevo ro es mayor (movimiento adelante)
        {
```



```
        n_dir_ro[0] = CW;
        n_ro[0] = r[0] - prev_r;
    }
    else
    {
        n_dir_ro[0] = CCW;
        n_ro[0] = prev_r - r[0];
    }

    // ir a lugar destino
    steps(n_ro[0], n_dir_ro[0], RO);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    steps(n_alpha[0], n_dir_alpha[0], ALPHA);
    // soltar
    gpio_set_level(iman_pin, 1);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    // regresar a posicion (0,0)
    calibrate();

    // liberar memoria
    heap_caps_free(n_dir_ro);
    heap_caps_free(n_dir_alpha);
    heap_caps_free(n_ro);
    heap_caps_free(n_alpha);

    cnt = 0;
}
}

void app_main()
{
    // void setup()
    init_hw();
    init_uart();
    // printf("Config Finalizada. \n");
    calibrate();

    int *length;

    while (1)
    {
        // asignar memoria
        uart_data = heap_caps_calloc(10, 1, MALLOC_CAP_8BIT);
        length = heap_caps_calloc(1, 1, MALLOC_CAP_8BIT);
    }
}
```

```
r = heap_caps_malloc(1, 1, MALLOC_CAP_8BIT);
a = heap_caps_malloc(1, 1, MALLOC_CAP_8BIT);

// leer uart
length[0] = uart_read_bytes(uart_num, uart_data, 12, 10);
if (length[0] > 0)
{
    // printf((const char *)uart_data);
    uart_flush(uart_num);

    if ((char)uart_data[0] == 'r')
    {
        get_uart_values(length[0]);
        rutina(length[0]); // funcion de rutina
    }
    else if ((char)uart_data[0] == 'f')
    {
        steps(1, CW, R0);
    }
    else if ((char)uart_data[0] == 'b')
    {
        steps(1, CCW, R0);
    }
    else if ((char)uart_data[0] == 'd')
    {
        steps(15, CCW, ALPHA);
    }
    else if ((char)uart_data[0] == 'l')
    {
        steps(15, CW, ALPHA);
    }
    else if ((char)uart_data[0] == 'c')
    {
        calibrate();
    }
}

// liberar memoria
heap_caps_free(uart_data);
heap_caps_free(length);
heap_caps_free(r);
heap_caps_free(a);

vTaskDelay(100 / portTICK_PERIOD_MS);
```

```
}  
}  
  
// IMPORTANTE - modificar sdkconfig.h #define CONFIG_FREERTOS_HZ 100 a 1000
```

Adicionalmente, se creó una GUI programada en Python, la cual maneja los datos ingresados por medio de la comunicación serial. El código de esta GUI es el siguiente:

```
import tkinter as tk  
from PIL import Image, ImageTk  
import serial  
import serial.tools.list_ports  
  
# seleccion automatica puerto serial de la esp  
ports = serial.tools.list_ports.comports()  
for onePort in ports:  
    if "Silicon Labs" in str(onePort):  
        com = str(onePort).split(' ')  
# config puerto serial  
s_port = serial.Serial()  
s_port.port = com[0]  
s_port.baudrate = 115200  
s_port.open()  
  
# funcion del boton Go  
def callback_go():  
    global cnt, s_port  
    s_port.flushInput()  
  
    send_text = 'r' + str(ro_slider.get()) + 'a' + str(alpha_slider.get()) +  
    ';'   
  
    s_port.write(send_text.encode('ascii'))  
    print(send_text)  
  
# callback control manual  
def callback_manual():  
    global control_manual, manual_button, up_button, down_button,  
    right_button, left_button  
    control_manual = not control_manual  
  
    if control_manual:  
        manual_button.config(bg='#282c34', fg="#FDFDFF")  
        up_button.config(state='active')
```

```
        down_button.config(state='active')
        right_button.config(state='active')
        left_button.config(state='active')
        calibrate_button.config(state='active')
        go.config(state='disabled')

    else:
        manual_button.config(bg="#FDFDFF", fg='#282c34')
        up_button.config(state='disabled')
        down_button.config(state='disabled')
        right_button.config(state='disabled')
        left_button.config(state='disabled')
        calibrate_button.config(state='disabled')
        go.config(state='active')

# callbacks botones control manual
def callback_up():
    global s_port
    s_port.write('f'.encode('ascii'))
    print('up')
def callback_down():
    global s_port
    s_port.write('b'.encode('ascii'))
    print('down')
def callback_right():
    global s_port
    s_port.write('d'.encode('ascii'))
    print('righth')
def callback_left():
    global s_port
    s_port.write('l'.encode('ascii'))
    print('left')

# callback para calibrar
def callback_calibrate():
    global s_port
    s_port.write('c'.encode('ascii'))
    print('calibrate')

# inicio de ventana
root = tk.Tk()
root.title("Brazo Robotico v1.0")
root.geometry(str(root.winfo_screenwidth()-50) + "x" +
str(root.winfo_screenheight()-10))
```

```
frame1 = tk.Frame(root, width=root.winfo_screenwidth(),
                  height=root.winfo_screenheight(), bg='#FDFDFF')
frame1.place(anchor='center', relx=0.5, rely=0.5)

# titulo
title_label = tk.Label(frame1, text="Brazo Robotico v1.0",
                       fg="#282c34", bg='#FDFDFF',
                       font=("System", '50', 'bold'),
                       justify='left')
title_label.place(relx=0.5, rely=0.1, anchor='center')

# alpha
alpha_img =
ImageTk.PhotoImage(Image.open('imag/transportador.png').resize((128,128)))
alpha_label_img = tk.Label(frame1, image=alpha_img, bg='#FDFDFF')
alpha_label_img.place(relx=0.31, rely=0.325, anchor='e')
alpha_label_text = tk.Label(frame1, text="Alpha",
                            fg="#282c34", bg='#FDFDFF',
                            font=("System", '32', 'bold'),
                            justify='left')
alpha_label_text.place(relx=0.35, rely=0.325, anchor='sw')
alpha_slider = tk.Scale(frame1, from_=0, to=180, orient='horizontal',
                        length=250, width=25, bg='#FDFDFF')
alpha_slider.place(relx=0.35, rely=0.325, anchor='nw')

# ro
ro_img = ImageTk.PhotoImage(Image.open('imag/scale.png').resize((128,128)))
ro_label_img = tk.Label(frame1, image=ro_img, bg='#FDFDFF')
ro_label_img.place(relx=0.31, rely=0.55, anchor='e')
ro_label_text = tk.Label(frame1, text="Ro",
                         fg="#282c34", bg='#FDFDFF',
                         font=("System", '32', 'bold'),
                         justify='left')
ro_label_text.place(relx=0.35, rely=0.55, anchor='sw')
ro_slider = tk.Scale(frame1, from_=0, to=250, orient='horizontal',
                    length=250, width=25, bg='#FDFDFF')
ro_slider.place(relx=0.35, rely=0.55, anchor='nw')

# boton de go
go = tk.Button(frame1, text='Go', width=10, height=2,
               fg="#FDFDFF", bg='#282c34',
               font=("System", '32', 'bold'),
               justify='center', command=callback_go)
```

```
go.place(relx=0.35, rely=0.775, anchor='center')

# control manual
# boton activar control manual
control_manual = False
manual_button = tk.Button(frame1, text="Control Manual",
                           bg="#FDFDFF", fg='#282c34',
                           font=("System", '32', 'bold'),
                           justify='center', command=callback_manual)
manual_button.place(relx=0.66, rely=0.25, anchor='center')
# up button
up_arrow =
ImageTk.PhotoImage(Image.open('imag/arrow.png').rotate(90).resize((64,64)))
up_button = tk.Button(frame1, image=up_arrow,
                      borderwidth=0, bg="#FDFDFF",
                      state='disabled', command=callback_up)
up_button.place(relx=0.66, rely=0.44, anchor='s')
# down button
down_arrow = ImageTk.PhotoImage(Image.open('imag/arrow.png').rotate(-
90).resize((64,64)))
down_button = tk.Button(frame1, image=down_arrow,
                        borderwidth=0, bg="#FDFDFF",
                        state='disabled', command=callback_down)
down_button.place(relx=0.66, rely=0.56, anchor='n')
# right button
right_arrow =
ImageTk.PhotoImage(Image.open('imag/arrow.png').resize((64,64)))
right_button = tk.Button(frame1, image=right_arrow,
                        borderwidth=0, bg="#FDFDFF",
                        state='disabled', command=callback_right)
right_button.place(relx=0.69, rely=0.50, anchor='w')
# left button
left_arrow =
ImageTk.PhotoImage(Image.open('imag/arrow.png').rotate(180).resize((64,64)))
left_button = tk.Button(frame1, image=left_arrow,
                        borderwidth=0, bg="#FDFDFF",
                        state='disabled', command=callback_left)
left_button.place(relx=0.63, rely=0.5, anchor='e')

# calibrate
calibrate_button = tk.Button(frame1, text="Calibrar",
                              width=10, height=2,
                              fg="#FDFDFF", bg='#282c34',
                              font=("System", '24', 'bold'),
```

```
justify='center', state='disabled',  
command=callback_calibrate)  
calibrate_button.place(relx=0.66, rely=0.775, anchor='center')  
  
root.mainloop()  
  
# cerrar puerto serial  
s_port.close()
```

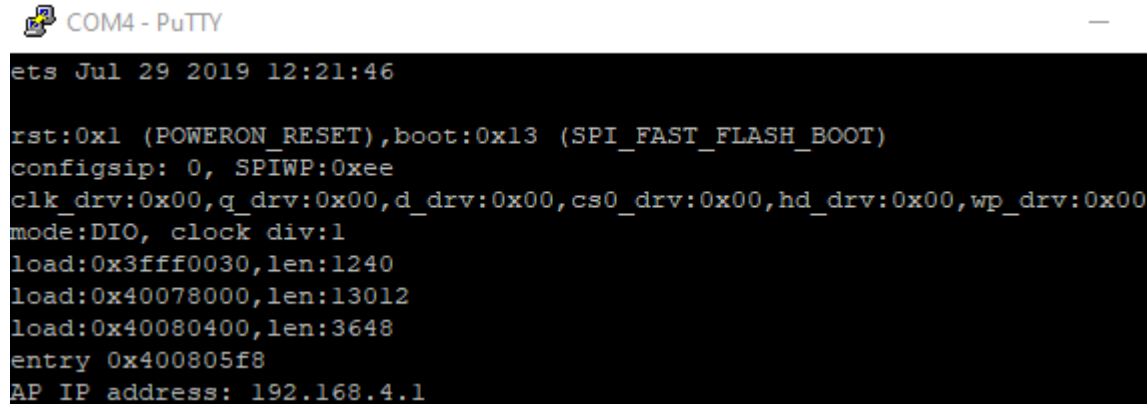


CÓDIGO ALTERNATIVO DE CONTROL

Las funciones necesarias que debía cumplir el brazo robótico consistían en controlar por medio de comunicación serial los datos ingresados mediante el usuario. Sin embargo, se optó por agregar de manera secundaria un sistema de comunicación mediante WiFi, ya que una de las principales características del microcontrolador ESP32, es que este tiene incluido una tarjeta WiFi y Bluetooth.

Este sistema de comunicación funciona habilitando la función de control WiFi por medio de un SoftAccessPoint, que nos permite hacer que se genere una red WiFi interna con un SSID definido, en el cual se puede acceder a un servidor local, diseñado en html, donde se puedan enviar datos directamente al microcontrolador mediante cualquier dispositivo conectado a la red WiFi generada.

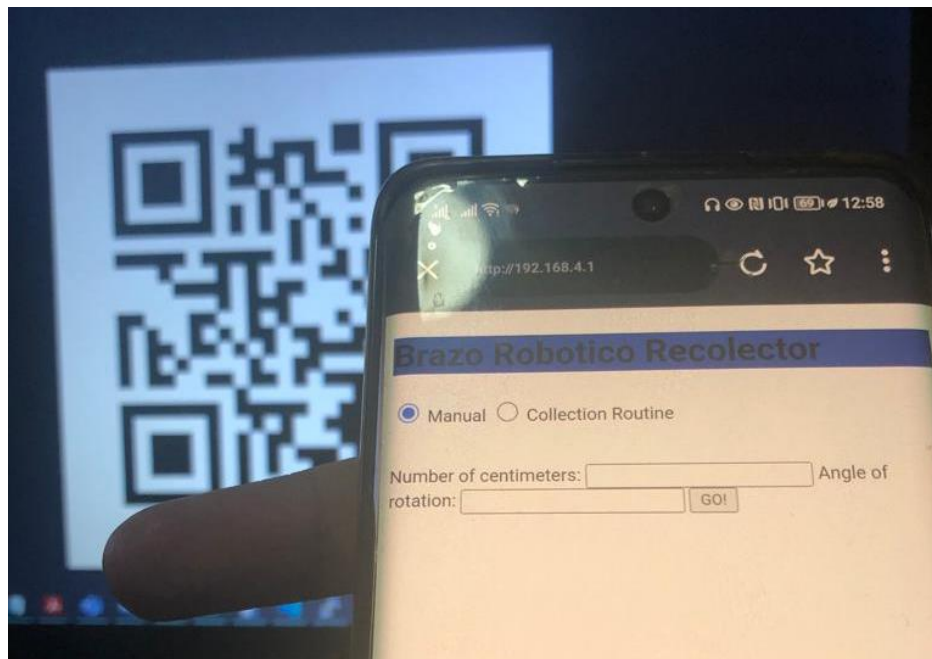
El servidor interno generado en la red WiFi se aloja en un IP definido que el microcontrolador proporciona por medio de la ventana de comunicación serial



```
ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1240
load:0x40078000,len:13012
load:0x40080400,len:3648
entry 0x400805f8
AP IP address: 192.168.4.1
```

Este IP se codifico en un código QR, que nos permite acceder al servidor local siempre y cuando estemos conectados a la red WiFi del SoftAP. Esta modificación nos permite tener versatilidad al momento de controlar el brazo robótico, ya que cualquier usuario puede enviar acciones al mismo solo conectándose a la red WiFi en un rango definido y accediendo a la dirección por medio del código QR.



Los datos enviados por medio de este servidor son utilizados para el control de recolección, mediante las rutinas habituales del código anterior, permitiendo el control manual adicionalmente.

El código para generar este servidor local es el siguiente:

```
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>

// Define stepper motor connections:
#define dirPinX 12
#define stepPinX 13
#define LSWX 32

#define dirPinA 26
#define stepPinA 27
#define LSWA 25

#define EI 33

int stepss = 0;
int ang = 0;
int c1=0;
int c2=0;
```

```
// Replace with your network credentials
const char* ssid = "BrazoRobotico";
const char* password = NULL;

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

// Search for parameters in HTTP POST request
const char* PARAM_INPUT_1 = "direction";
const char* PARAM_INPUT_2 = "steps";
const char* PARAM_INPUT_3 = "angle";

// Variables to save values from HTML form
String direction;
String steps;
String angle;

// Variable to detect whether a new request occurred
bool newRequest = false;

// HTML to build the web page
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE html>
<html>
<head>
  <title>Stepper Motor</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
```

```
<body>
<h1 style="background-color:rgb(0, 75, 168);">Brazo Robotico Recolector</h1>
<form action="/" method="POST">
  <input type="radio" name="direction" value="CW" checked>
  <label for="CW">Manual</label>
  <input type="radio" name="direction" value="CCW">
  <label for="CW">Collection Routine</label><br><br><br>
  <label for="steps">Number of centimeters:</label>
  <input type="number" name="steps">
  </label for="angle">Angle of rotation:</label>
  <input type="number" name="angle">
  <input type="submit" value="GO!">
</form>
</body>
</html>
)rawliteral";
```

```
// Initial ize WiFi
void initWiFi() {
  WiFi.mode(WIFI_AP);
  WiFi.softAP(ssid, password);
  IPAddress IP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(IP);
}

void electmag(byte state){
  if(state == 0){
```

```
    digitalWrite(EI, LOW);
}
if(state == 1){
    digitalWrite(EI, HIGH);
}
}

void motorX(float dist, int direccion)
{
    stepss=1000*dist;

    if (direccion == 1) //Direccion positiva
    {
        digitalWrite(dirPinX, HIGH);
    }
    else if (direccion == 0) //Direccion negativa
    {
        digitalWrite(dirPinX, LOW);
    }
    for(int i = 0; i < stepss; i++)
    {
        //1kHz wave
        digitalWrite(stepPinX, HIGH);
        delayMicroseconds(500);
        digitalWrite(stepPinX, LOW);
        delayMicroseconds(500);
    }
}
```

```
void motorA(float dist, int direccion)
{
    ang=(3200/360)*dist;

    if (direccion == 1) //Direccion positiva
    {
        digitalWrite(dirPinA, HIGH);
    }
    else if (direccion == 0) //Direccion negativa
    {
        digitalWrite(dirPinA, LOW);
    }

    for(int i = 0; i < ang; i++)
    {
        //1kHz wave
        digitalWrite(stepPinA, HIGH);
        delayMicroseconds(500);
        digitalWrite(stepPinA, LOW);
        delayMicroseconds(500);
    }
}

void setup() {
    // Declare pins as output:
```

```
pinMode(stepPinX, OUTPUT);
pinMode(dirPinX, OUTPUT);
pinMode(LSWX, INPUT);

pinMode(stepPinA, OUTPUT);
pinMode(dirPinA, OUTPUT);
pinMode(LSWA, INPUT);

pinMode(EI, OUTPUT);

//Monitor serial
Serial.begin(115200);
initWiFi();

// Web Server Root URL
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/html", index_html);
});

// Handle request (form)
server.on("/", HTTP_POST, [](AsyncWebServerRequest *request) {
    int params = request->params();
    for(int i=0;i<params;i++){
        AsyncWebParameter* p = request->getParam(i);
        if(p->isPost()){
            // HTTP POST input1 value (direction)
            if (p->name() == PARAM_INPUT_1) {
                direction = p->value().c_str();
            }
        }
    }
}
```

```
    Serial.print("Direction set to: ");
    Serial.println(direction);
}

// HTTP POST input2 value (steps)
if (p->name() == PARAM_INPUT_2) {
    steps = p->value().c_str();
    Serial.print("Number of steps set to: ");
    Serial.println(steps);
}

// HTTP POST input3 value (steps)
if (p->name() == PARAM_INPUT_3) {
    steps = p->value().c_str();
    Serial.print("Angle of rotation set to: ");
    Serial.println(angle);
}
}
}

request->send(200, "text/html", index_html);
newRequest = true;
});

server.begin();
}

void gethome(){
    c1=1;
    c2=1;
    while((c1==1) or (c2 == 1)) {
```

```
if(digitalRead(LSWA)==HIGH){
    motorA(0.1,0);
}
else{
    c1=0;
}
}

if(digitalRead(LSWX)==LOW){
    motorX(0.1,0);
}
else{
    c2=0;
}
}

void loop() {
    // Check if there was a new request and move the stepper accordingly
    if (newRequest){

        if (direction == "CW"){
            if(steps.toInt()<0){
                motorX(abs(steps.toInt()),0);
            }
            else{
                motorX(abs(steps.toInt()),1);
            }
        }
        else{
```



```
//Desplaza el riel en la direccion positiva
motorX(steps.toInt(),1);
delay(1000);

//Rota el brazo en la direccion positiva
motorA(angle.toInt(),1);
delay(1000);

//Enciende el electroiman
electmag(1);
delay(100);

//Rota el brazo en la direccion positiva
motorA(angle.toInt(),0);
delay(1000);

//Desplaza el riel en la direccion positiva
motorX(steps.toInt(),0);
delay(1000);

//Enciende el electroiman
electmag(0);
delay(2000);
newRequest = false;
}
}
}
```

PROBLEMAS PRESENTADOS EN EL PROYECTO

El primer problema presentado fue el control de los motores mediante la tarjeta ESP32 en el entorno de programación de Espressif, ya que la velocidad de desplazamiento del motor era muy baja debido a que estábamos utilizando una función incorrecta para la generación del tren de pulsos encargado de darle la secuencia de encendido a las bobinas de los motores de pasos. Para solucionar esto, nuestra primera forma de verificar cual era el problema realmente fue conectando un osciloscopio al pin de salida del ESP32 correspondiente a la generación del tren de pulsos. En este pin nos dimos cuenta de que no se estaba generando la señal deseada, por lo que se varió la frecuencia hasta determinar una en la cual si se generara el tren de pulsos pero esta seguía siendo muy baja, por lo que se entró a la librería de sdkconfig.h #define CONFIG_FREERTOS_HZ de 100 a 1000 Hz.

El segundo problema más grande que tuvimos fue la implementación mecánica, ya que había que diseñar una forma de hacer rotar un eje de un peso significativo, el cual podía resistir al torque máximo del motor, por lo que nos aseguramos de disminuir al máximo fricción, agregando aceite por donde se desplazan las ruedas locas y una tira de tape.

Mas adelante, seguíamos teniendo problemas con el torque suministrado por el motor, por lo que levantamos ligeramente el eje del riel, ya que este presentaba una pequeña inclinación con respecto a la perpendicularidad del eje del motor, produciendo una torsión en el eje del motor en ciertos puntos del recorrido.

El ultimo inconveniente presentado fue la perdida de una de las bobinas del motor de pasos del movimiento lineal, debido a la alta carga producida por la fricción y a la torsión producida por desalineamiento. Este error fue solucionado disminuyendo al mínimo las posibles causas de fricción y torsión, además de agregar un pequeño factor de corrección dentro del algoritmo que traduzca la nueva cantidad de pasos que generan una cantidad específica de ángulos.

CONCLUSIÓN

Teniendo en cuenta todos los resultados que se han mostrado durante el presente proyecto, a continuación, se establecerán las conclusiones de esta experiencia desarrollando el brazo robótico durante el curso.

1. Tuvimos la oportunidad de desarrollar un brazo robótico de manera colaborativa que incluyo el diseño e impresión 3D de la estructura (conexiones y piezas), dimensionamiento y selección de circuitos y/o componentes electrónicos, programación y control.
2. Uno de los inconvenientes de comprar las cosas ya hechas, por ejemplo, en este caso un brazo robótico, es que te pierdes de la oportunidad y la experiencia de diseñar, armar, ensamblar etc... en fin todo lo que involucra el aprendizaje de los procesos mecánicos, eléctricos y de programación. Por lo tanto, es de primera importancia realizar este tipo de trabajos de donde se desarrolla un proyecto desde 0 de manera grupal, ya que les brinda a los estudiantes no solo la oportunidad de trabajar en equipo para lograr un objetivo en común que es una buena simulación a lo que estaremos haciendo cuando vayamos a trabajar en una empresa sino también la experiencia práctica obtenida que conlleva el desarrollo de este tipo de proyectos.
3. Este proyecto fue bastante complejo de desarrollar debido a los fuertes conocimientos prácticos, de control y programación requeridos para llevarlo a cabo, y resultó ser un gran desafío su construcción.
4. Consideramos que como grupo hicimos un excelente trabajo porque, aunque tuvimos obstáculos e inconvenientes durante la realización del proyecto, fuimos capaces de desarrollar un brazo robótico de bajo costo funcional con las especificaciones pedidas por el profesor que era al final el propósito del proyecto. Puede que no tengamos un brazo robótico de 1000 dólares, pero estamos satisfechos con lo logrado.
5. Vivimos en un mundo donde la tecnología avanza día a día y la programación es una de las mejores herramientas del ingeniero eléctrico por lo cual familiarizarse con la programación del ESP32 es de vital importancia para todo profesional de la ingeniería que quiera mantenerse actualizado ante el inminente cambio digital.

Debido a la magnitud y complejidad del proyecto, algunas ideas no fueron exploradas. Por lo tanto, a continuación, se listan una lista de sugerencias que podrían ser incluidas en el mejoramiento del proyecto:

1. Se sugiere hacer algunos cambios en la estructura con respecto al aspecto de construcción del brazo, de manera que favorezca y aligera la presión y fricción a la que es sometida el motor, reforzar la estructura para soportar más peso, aumentar la potencia de los motores.

REFERENCIAS

- [1] A. S. Sedra y K. C. Smith. *Circuitos Microelectrónicos*: Cuarta Edición, México, DF, Ed Oxford, 1999.
- [2] Floyd, Thomas L. Dispositivos Electrónicos. 8a. ed. México: Pearson Education, 2008.
- [3] Hart, Daniel. *Electrónica de Potencia*. Prentice Hall. 2001.
- [4] R. L. Boylestad, *Electrónica: Teoría de Circuitos*, México, DF, Ed. Prentice Hall, 1997.
- [5] Cameron, Neil. *Electronics Projects with the ESP8266 and ESP32*. Berkeley, CA: Apress, 2021. <http://dx.doi.org/10.1007/978-1-4842-6336-5>.
- [6] Cameron, Neil. "ESP32 microcontroller features." In *Electronics Projects with the ESP8266 and ESP32*, 641–82. Berkeley, CA: Apress, 2020. http://dx.doi.org/10.1007/978-1-4842-6336-5_22.
- [7] Espressif Systems ESP 32 Series of Modules. Recuperado de <https://www.espressif.com/en/products/modules/esp32>.