Carpeta Técnica

Wall-H

INDICE:

1. IDENTIFICACION..... pag 5

1.1 NOMBRE DEL PROYECTO ..... pag 5
1.2 INTEGRANTES..... pag 5
1.3 FOTO GRUPAL..... pag 6
1.4 DOCENTES RESPONSABLES..... pag 6
1.5 FECHA DE INICIO..... pag 6
1.6 ESFUERZO DEL PROYECTO..... Pag 6
1.7 TAREAS ABARCADAS..... pag 6

2. INFORMACION GENERAL..... pag 7

1.0.1 OBJETIVO..... pag 7
1.0.2 USOS DEL PROYECTO..... Pag 7
1.0.3 DESCRIPCION DEL FUNCIONAMIENTO..... pag 7
1.0.4 CARACTERISTICAS GENERALES..... Pag 8

1.1 ALCANCE SOCIAL Y GEOGRAFICO..... pag 8

2.0 ESTRUCTURA Y MOTORES..... pag 9

2.0.1 DESCRIPCION Y VENTAJAS..... pag 9
2.0.2 CARACTERISTICAS TECNICAS..... pag 9
2.0.3 DISEÑO..... pag 10

- VISTA LATERAL..... Pag 11
- VISTA FRONTAL..... Pag 12
- VISTA SUPERIOR..... pag 13
- VISTA INFERIOR..... Pag 14
- VISTA INFERIOR..... Pag 15
- DESPIECE..... pag 16

2.0.4 PLANOS..... pag 17

- BASE..... pag 17
- GRAMPA PARA BATERIAS..... Pag 18
- GRAMPA..... pag 19
- PERFIL DE ALUMINIO CUADRADO..... pag 20
- PIEZA DENTADA..... pag 21
- POLEA..... pag 22
- PUNTERA DE BASE..... Pag 23
- UNION POLEAS..... Pag 24

2.1 MOTORES..... pag 25
2.1.1 DESCRIPCION GENERAL..... pag 25
2.1.2 CARACTERISTICAS TECNICAS..... pag 25

3.0 MODULO CONTROLADOR DE MOTORES..... pag 25
3.0.1 DESCRIPCION GENERAL..... pag 25
3.0.2 CARACTERISTICAS TECNICAS..... Pag 26
3.0.3 DIAGRAMA EN BLOQUES..... pag 27

3.1 CONTROL LOGICO..... pag 27
3.1.1 DESCRIPCION..... pag 27
3.1.2 BATERIA DE 5V..... pag 27
3.1.3 CONTROL DE ENERGIA..... Pag 27

4.0 SISTEMA DE CONTROL DEL ROBOT..... pag 28
4.0.1 DESCRIPCION Y FUNCIONES DEL MODULO ULTRASONICO..... pag 28
4.0.2 DIAGRIAMA EN BLOQUES..... pag 28

4.1 SENSORES ULTRASONICOS..... pag 28
4.1.1 FUNCIONAMIENTO..... pag 28
4.1.2 MEDICION..... pag 29
4.1.3 PROGRAMACIÓN..... Pag 29
4.1.4 ESQUEMATICO..... Pag 30

4.2 MODULO DE CAMARA..... pag 30
4.2.1 DESCRIPCION..... pag 30
4.2.2 UTILIZACION..... pag 30
4.2.3 PROGRAMACION..... pag 30

4.3 JOYSTICK DE CONTROL..... pag 44
4.3.1 DESCRIPCION..... pag 44
4.3.2 UTILIZACION..... pag 44
4.3.3 PROGRAMACION..... pag 44

4.4 CONTROL CON EL PUENTE H..... pag 46
4.4.1 DESCRIPCION..... pag 46
4.4.2 UTILIZACION..... pag 46
4.4.3 PROGRAMACION..... pag 46

5.0 SISTEMA DE BATERIAS..... Pag 50
5.0.1 DESCRIPCION..... pag 50
6.0 SISTEMA DE COMUNICACIONES..... pag 50
6.0.1 INTRODUCCION..... Pag 50
6.1 SISTEMA DE COMUNICACIONES INTERNOS..... pag 51
6.1.1 BUS I2C..... pag 50
6.2 SISTEMA DE COMUNICACIÓN CON EQUIPOS EXTERNOS..... Pag 52
6.2.1 ENLACE WIFI..... pag 52
6.3 COMUNICACIÓN DEL SERVIDOR DE CONTROL..... pag 52
6.3.1 DESCUBRIMIENTO DEL SERVIDOR..... pag 52
6.3.2 SISTEMA DE PROTOCOLOS..... pag 53
6.3.3 COMUNICACIÓN SERVIDOR-CLIENTE..... pag 53
7.0 INTERFAZ DE USUARIO..... Pag 54
8.0 MODULO DE MEDICION DE SIGNOS VITALES..... Pag 54
8.0.1 DESCRIPCION..... pag 54
8.1 SENSOR DE TEMPERATURA..... Pag 54
8.1.1 DESCRIPCION..... Pag 54
8.1.2 DIAGRAMA DE CONEXIÓN..... Pag 54
8.1.3 FUNCIONAMIENTO..... pag 55
8.1.4 CARACTERISTICAS..... pag 55
8.1.5 PROGRAMACION..... pag 56
8.1.6 ESQUEMATICO..... pag 57
8.2 SENSOR OXIMETRO..... pag 58
8.2.1 DESCRIPCION..... Pag 58
8.2.2 DIAGRAMA DE CONEXIÓN..... Pag 58
8.2.3 FUNCIONAMIENTO..... pag 58
8.2.4 CARACTERISTICAS..... Pag 59
8.2.5 PROGRAMACION..... Pag 60
8.2.6 ESQUEMATICO..... Pag 61
8.3 DISPLAY OLED..... pag 61
8.3.1 DESCRIPCION..... pag 62

9.0 MODULO MEDIDOR DE DIOXIDO DE CARBONO…… pag 62
9.0.1 DESCRIPCION…… Pag 62
9.1 SENSOR DE CALIDAD DE AIRE…… pag 62
9.1.1 DESCRIPCION…… Pag 62
9.1.2 DIAGRAMA DE CONEXIÓN…… Pag 63
9.1.3 CARACTERISTICAS…… Pag 63
9.1.4 FUNCIONAMIENTO…… pag 63
9.1.5 PROGRAMACION…… Pag 64
9.1.6 ESQUEMATICO…… Pag 65
10.0 INFORMACION DE TRABAJO…… pag 66
10.1 CONDICIONES DE ENTORNO…… pag 66
10.2 INSTRUMENTAL UTILIZADO…… pag 66
10.3 BIBLIOGRAFIA…… pag 66

0.0 Identificación:

0.1.0 Nombre del proyecto: Wall-H

0.2.0 Integrantes:

- Bourlot, David. DNI: 44.707.701 7mo 2da aviónica
- Flores, Geraldine. DNI: 44.506.964 7mo 2da aviónica
- Fotanazzi, Valentino. DNI: 44.958.837 7mo 2da aviónica
- Montoni, Juan Manuel. DNI: 44.958.843 7mo 2da aviónica
- Moreno, Nicolás. DNI: 44.264.449 7mo 2da aviónica

0.3.0 Foto Grupal:

Imagen 1.0- Foto grupal

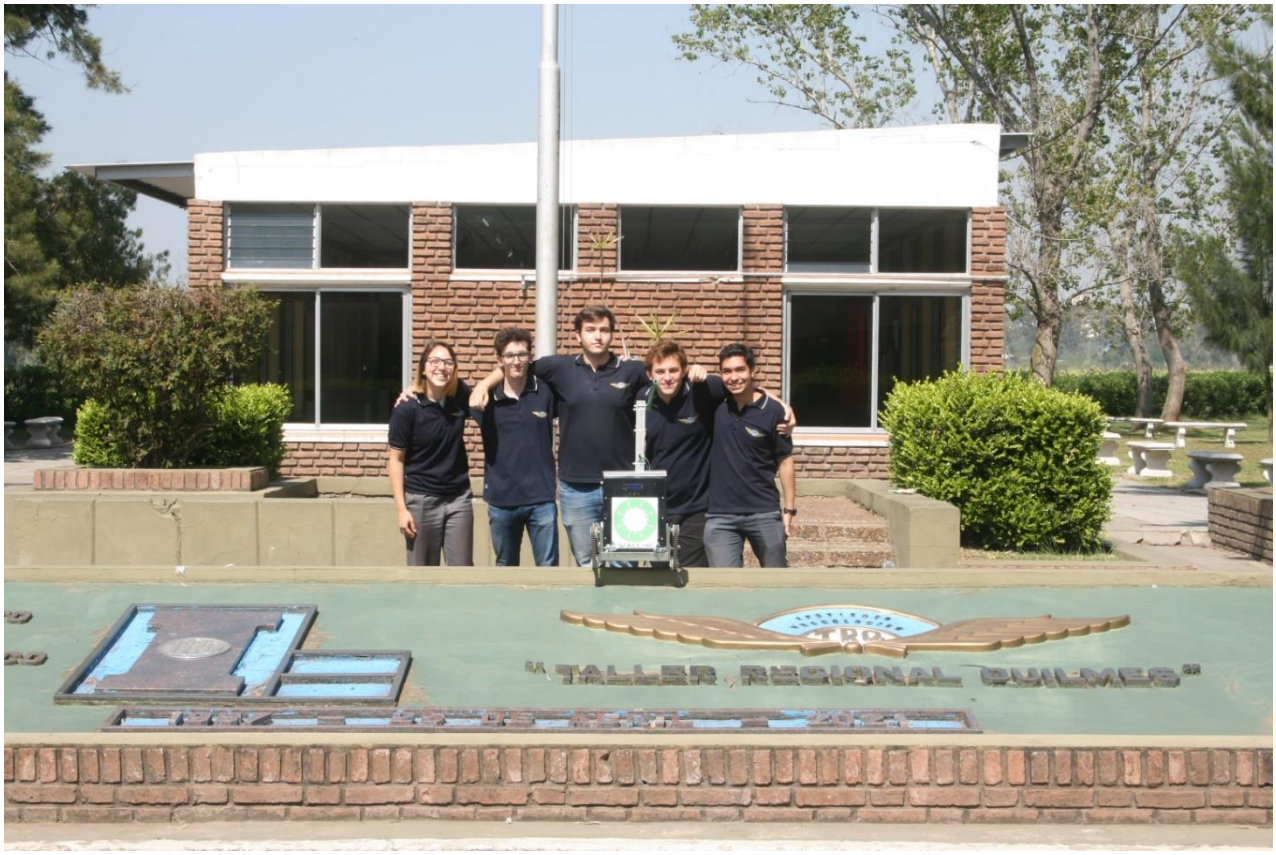


Imagen 2 - Foto grupal

0.4 Docentes responsables:

- Züge, Magali
- Medina, Sergio
- Alegre, Marcos
- Bianco Cesar Carlos
- Carlassara, Fabricio

0.5 Fecha de inicio: 10/10/2020

0.6 Esfuerzo del proyecto:

- 60 semanas de trabajo
-1440 horas de trabajo
- 24 horas semanales repartidas entre 5 integrantes

0.7 Tareas abarcadas:

- Investigación de problemática
-Desarrollo del producto
- Difusión y establecimiento en redes sociales
-Creación de página web

1.0 Información general:

1.0.1 Objetivo:

Desarrollamos y ensamblamos una unidad automatizada para reducir el contacto directo del paciente con sintomatologías infecciosas y el personal de medicina, mediante un desarrollo de electrónica/informático. Este mide el ritmo cardiaco, temperatura, oxígeno en sangre para con estos datos, poder controlar la condición de los pacientes, y generar un registro en el sistema.

1.0.2 Usos del proyecto:

Los médicos y enfermeros evitarán el contacto constante con el paciente, va a tomar la frecuencia cardiaca, temperatura y oxígeno en sangre del paciente de forma más rápida y eficaz.

Muestra los datos recaudados del paciente con mayor eficacia al enviarlos automáticamente a una base de datos en el ordenador del médico, unificando la ubicación de la información.

1.0.3 Descripción del funcionamiento:

La unidad Robótica es capaz de medir diferentes valores por medio de del sensor oxímetro y el termómetro infrarrojo (ritmo cardiaco, temperatura, oxígeno en sangre) para mostrarle los valores de dichas mediciones al personal médico. El robot se va a mover mediante un Joystick que estará en un aparato de control del robot, esté siendo capaz de moverse hacia el frente o girar 90°. Su movimiento es libre. La unidad la vamos a usar en el ala de internación

1.0.4 Características generales:

Item	Valor	Unidad
Ancho	38	Cm
Largo	61	Cm
Altura	80	Cm
Peso	5	Kg
Rango de comunicación WiFi	100	m
Tiempo de carga	12	h
Velocidad máxima	0,15	m/s
Tensión de trabajo	12	V
Area de trabajo	10	Km
Máxima pendiente	60	°
Temperatura de uso	80	º

Tabla 1- Medidas del robot

1.1 Alcance social y geográfico:

Empezamos a hacer este proyecto el 30 de octubre del 2020, año en el cual se desato la pandemia del Covid-19, esto nos hizo plantear que necesitábamos ser parte de una posible solución. Gracias a este planteo se nos ocurrió diseñar una unidad robótica la cual ayudaría a el personal de la salud, que es esta época de pandemia es el más afectado, y basándonos en estudios que encontramos en internet proporcionado la página del Ministerio de Salud de la Nación una estadística que nos sorprendió, que es la cantidad de contagios que se dan por el contacto estrecho (imagen 4).

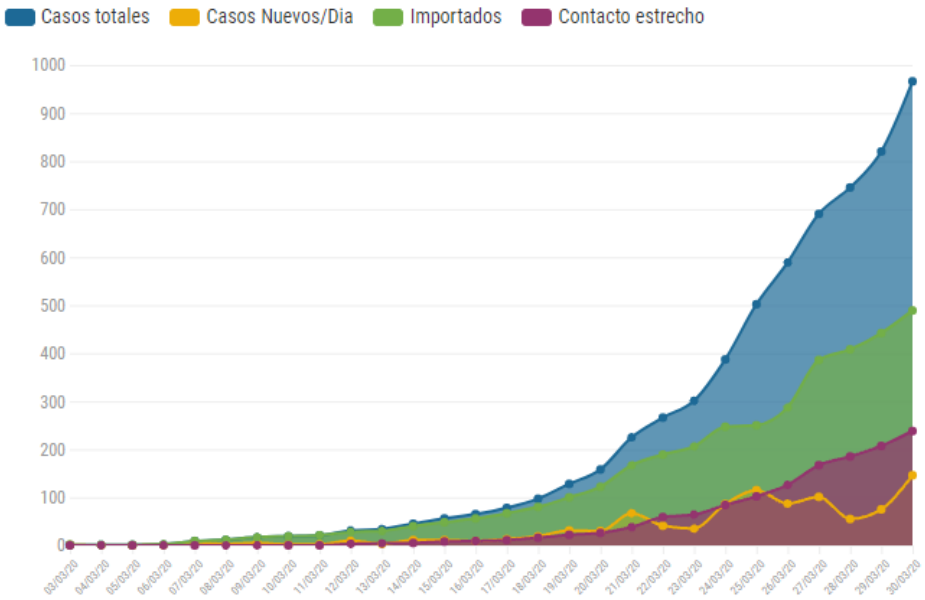


Imagen 3- gráfico de casos positivos por COVID 19

Gracias a esta información que encontramos nos planteas diseñar esta unidad robótica la cual ayudaría a que este contacto estrecho entre paciente y personal médico no se dé, y que se reduzcan la cantidad de casos positivos por covid-19

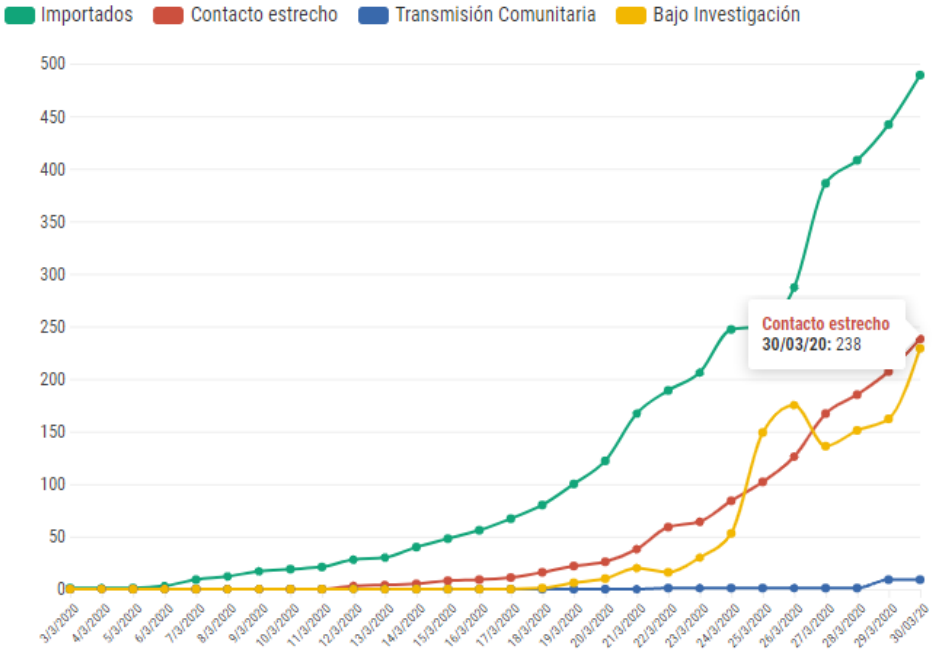


Imagen 4- gráfico de casos positivos por COVID 19 238 casos

2.0 Estructura y motores:

2.0.1 Descripción y ventajas:

La estructura del Wall-H esta diseñada para ser resistente a impactos, rigida pero versatil en distintos terrenos, los materiales usados para su construccion son inoxidable, aluminio, PLA y hierro, la traccion de Wall-H es diferencial y se realiza a traves de dos orugas con motores individuales, esto le da un exelente agarre a distintos suelos ademas de darle agilidad al trasladarse por terrenos irregulares.

2.0.2 Carcterísticas técnicas:

Tabla 2- Peso del robot

Item	Valor	Unidad
Peso sin baterías	4	Kg
Peso con baterías	10	Kg
Carga máxima	20	Kg

2.0.3 Diseño:

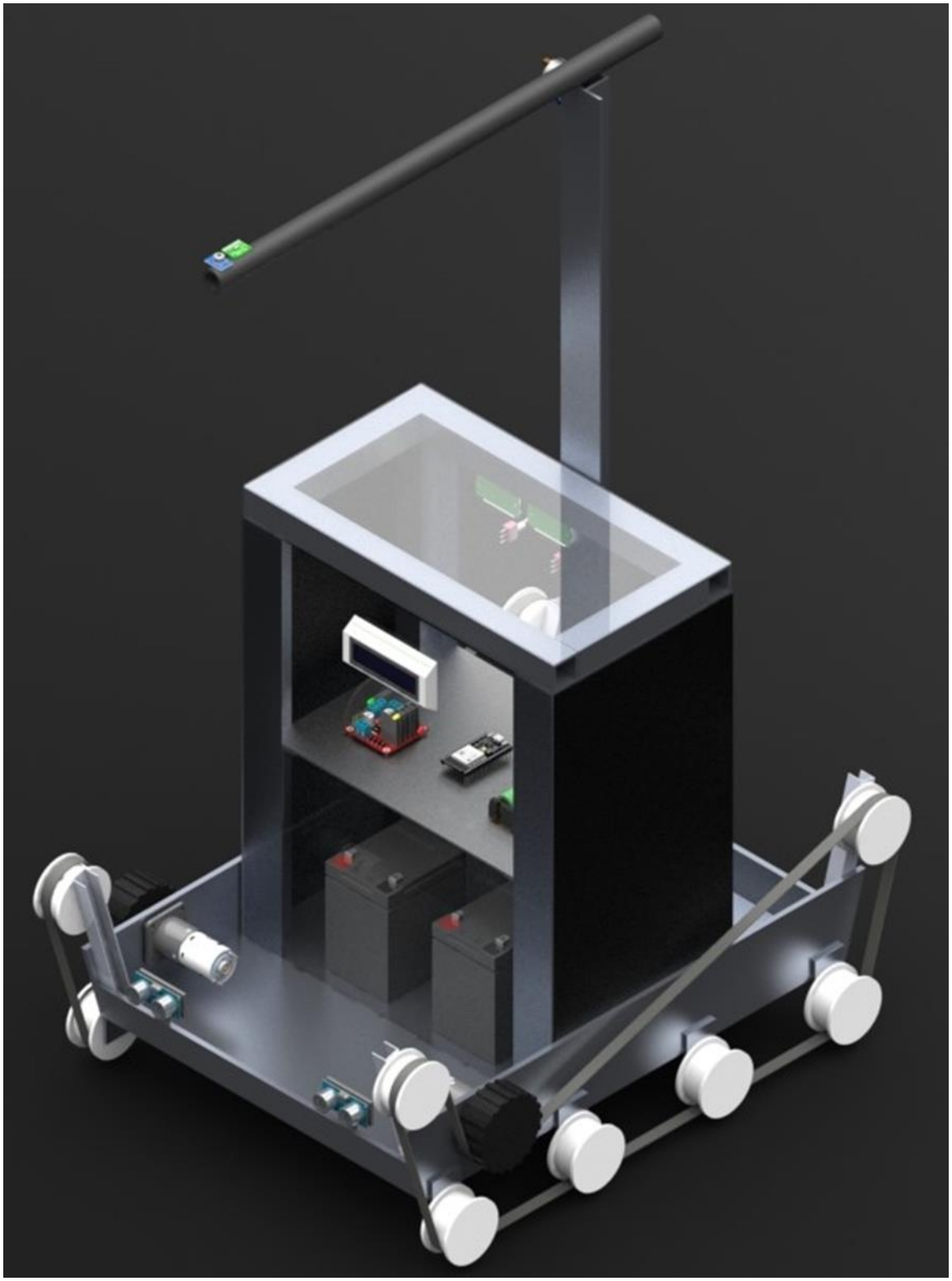


Imagen 5- Diseño del robot

- Vista lateral:

Imagen 6- Diseño del robot vista lateral

- Vista frontal:

Imagen 7- Diseño del robot vista frontal

- Vista superior:

Imagen 8- Diseño del robot vista superior

- Vista trasera:

Imagen 9- Diseño del robot vista trasera

- Vista inferior:

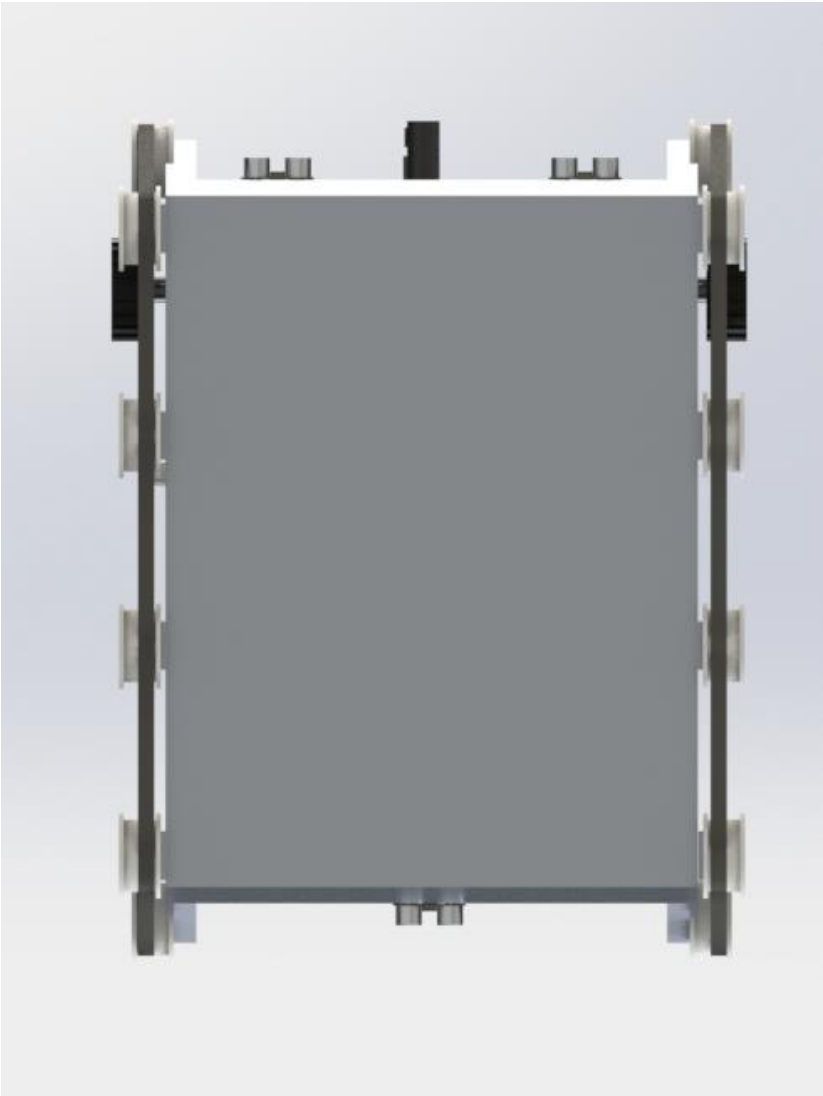


Imagen 10- Diseño del robot vista inferiror

- Despiece:

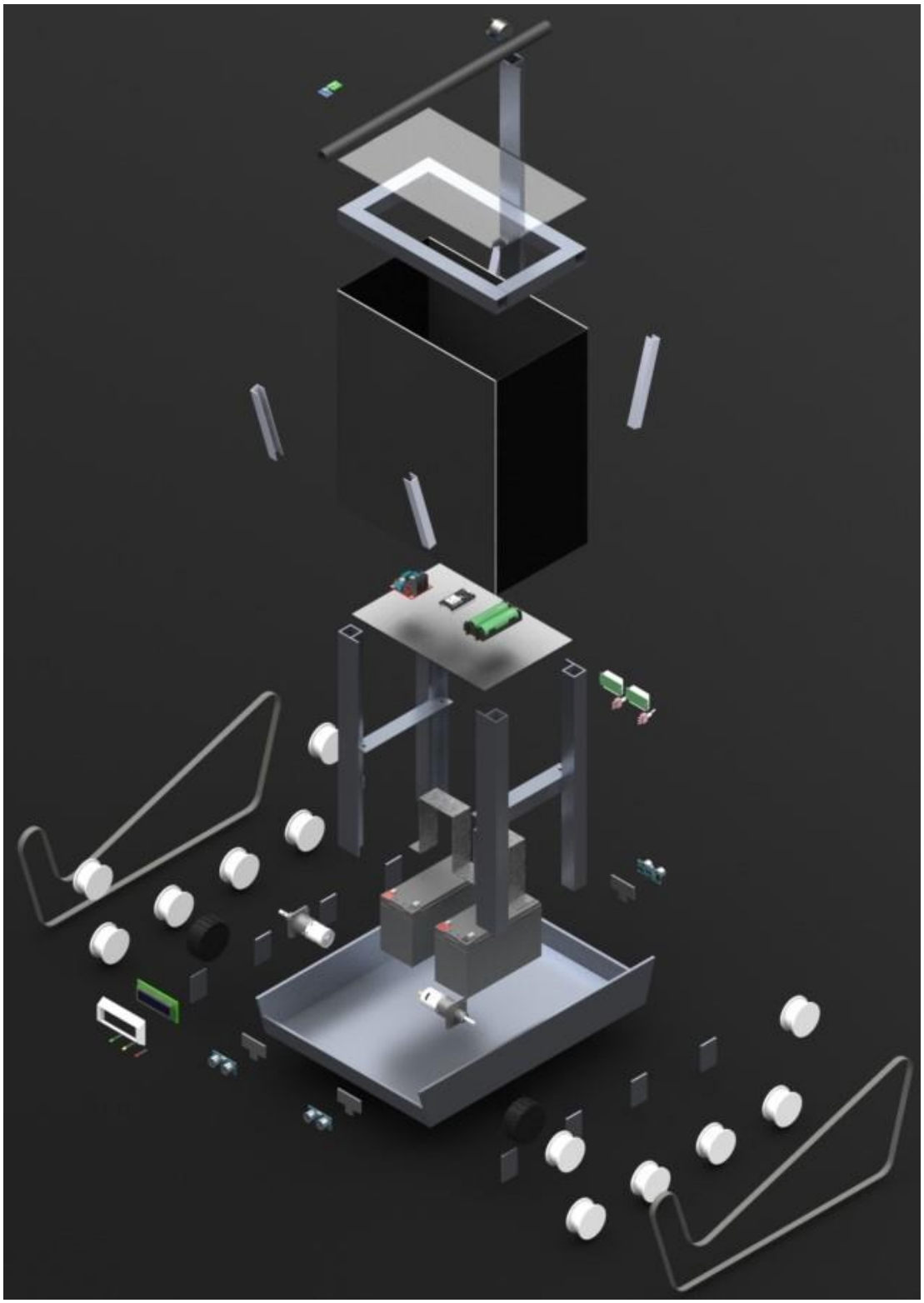


Imagen 11- Diseño del robot despiece

2.0.4 Planos:

- Base:

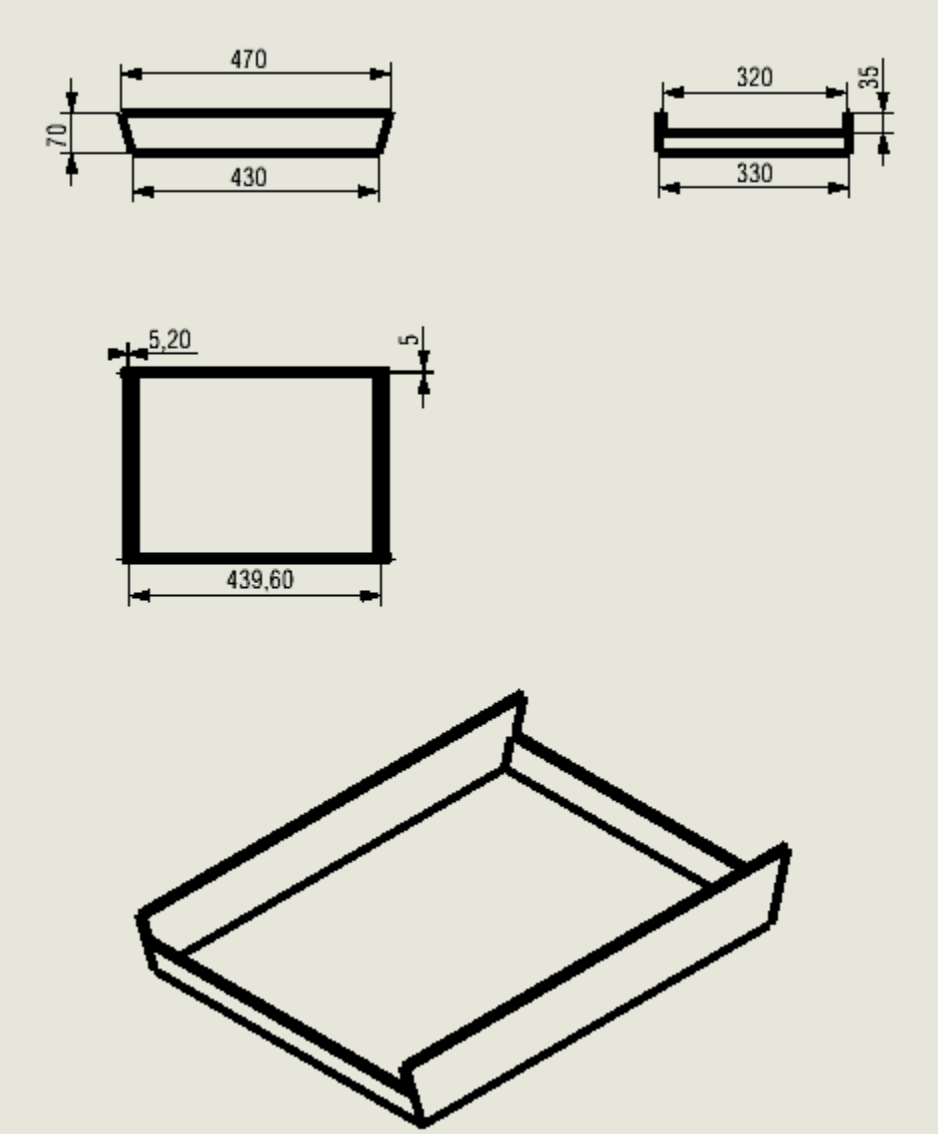


Imagen 12- Plano de la base del robot

- Grampa de las baterías:

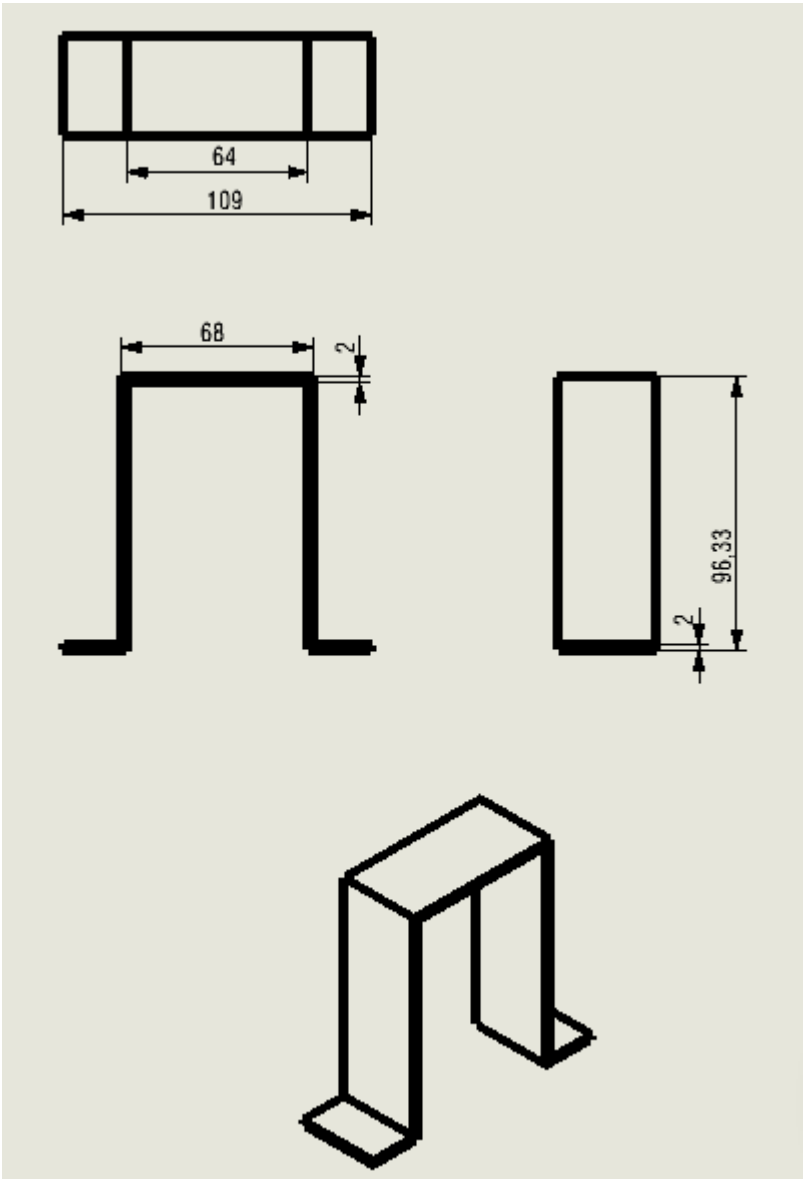


Imagen 13- Plano de la grampa de las baterías

- Grampa:

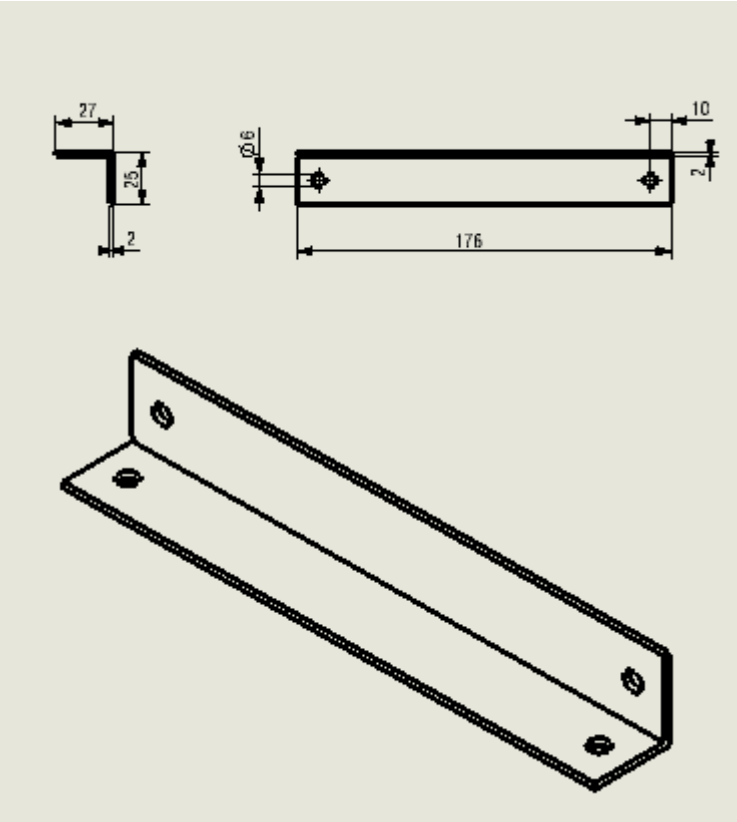


Imagen 14- Plano de las grampas del robot

- Perfil de aluminio cuadrado:

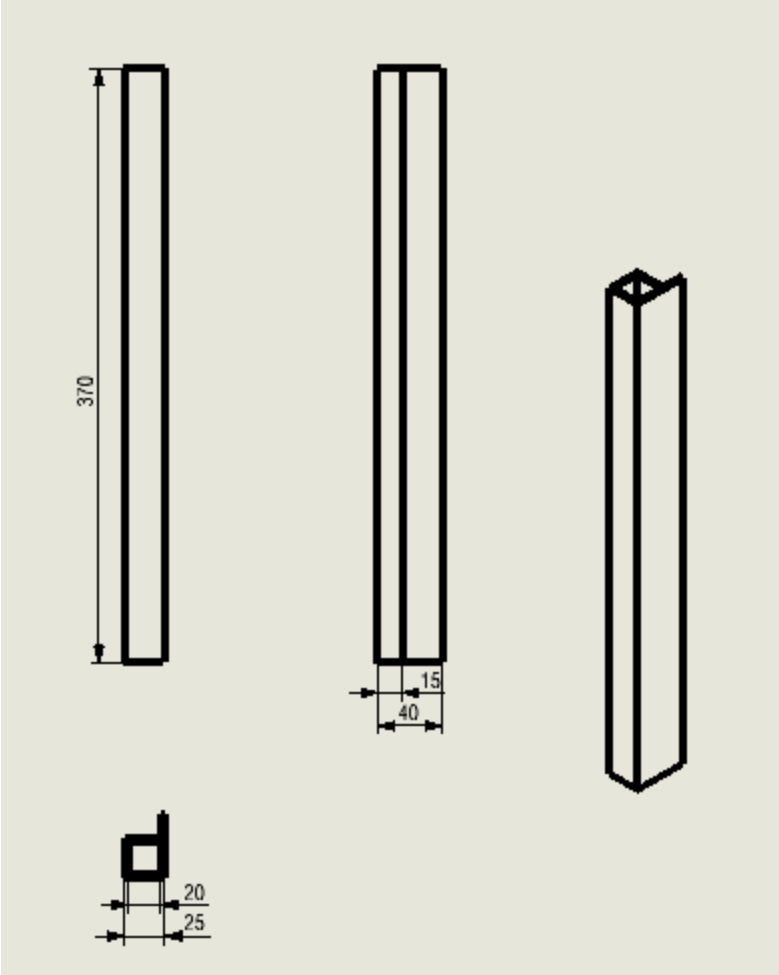


Imagen 15- Plano de los perfiles de aluminio cuadrado

- Pieza dentada:

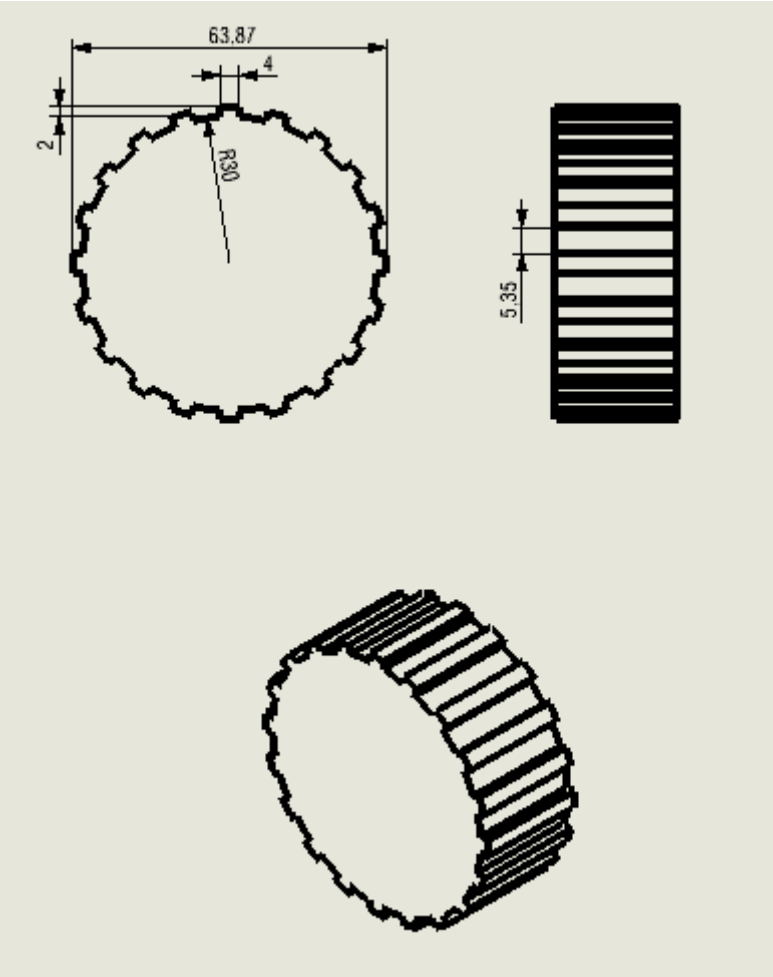


Imagen 16- Plano de las piezas dentado

- Polea:

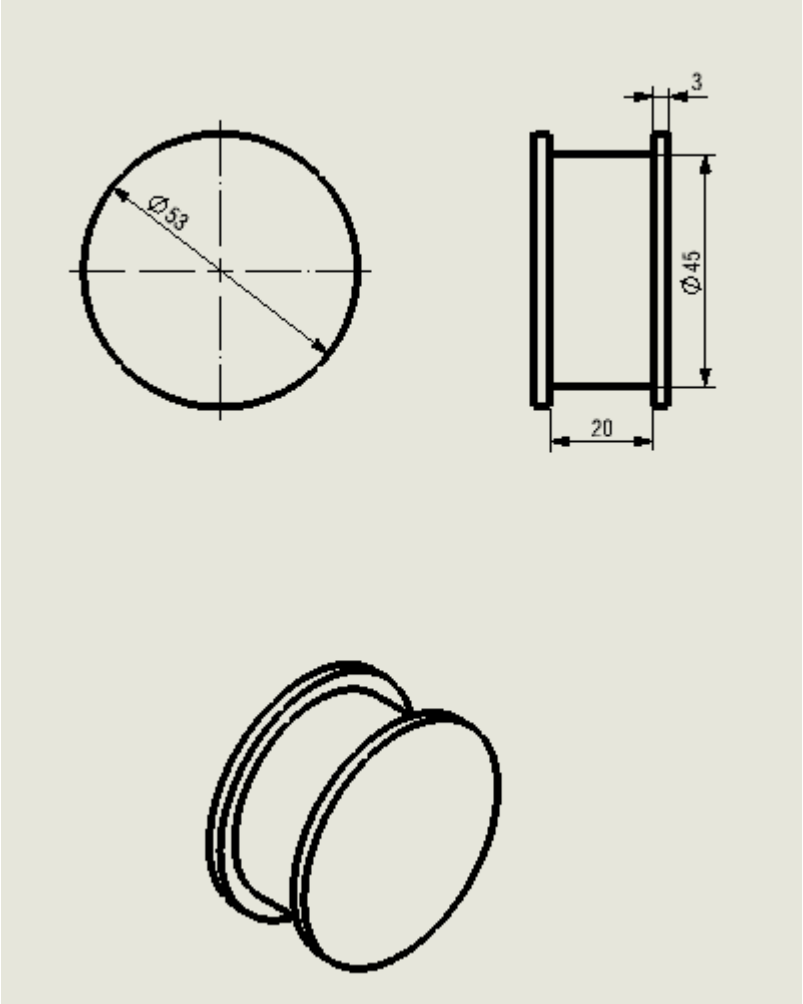


Imagen 17- Plano de las poleas

- Punteras de base:

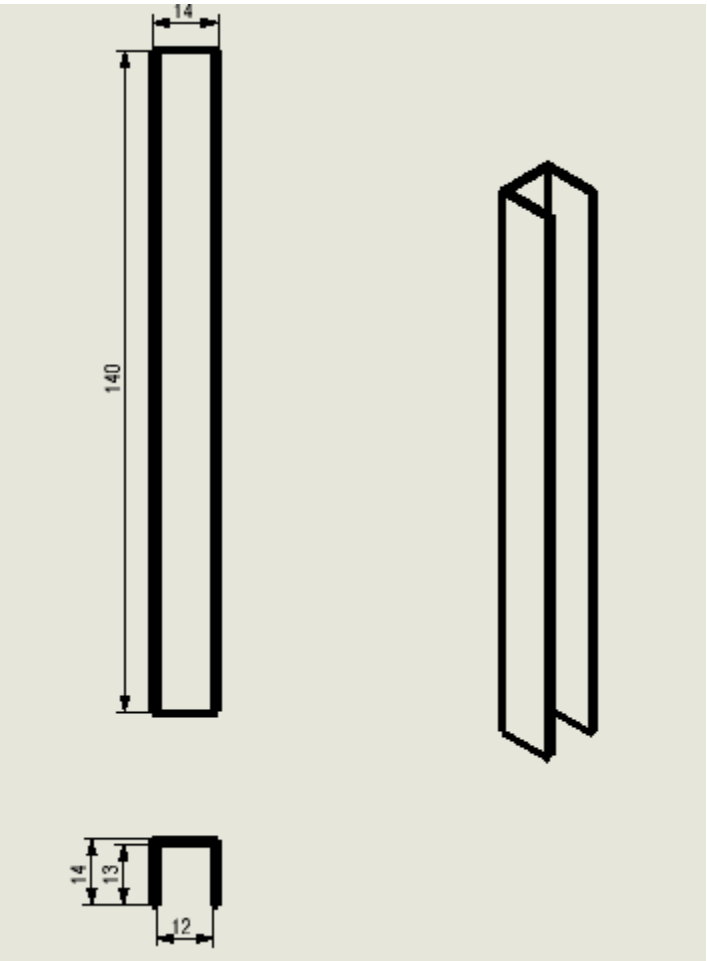


Imagen 18- Plano de punteras de base

- Unión de poleas:

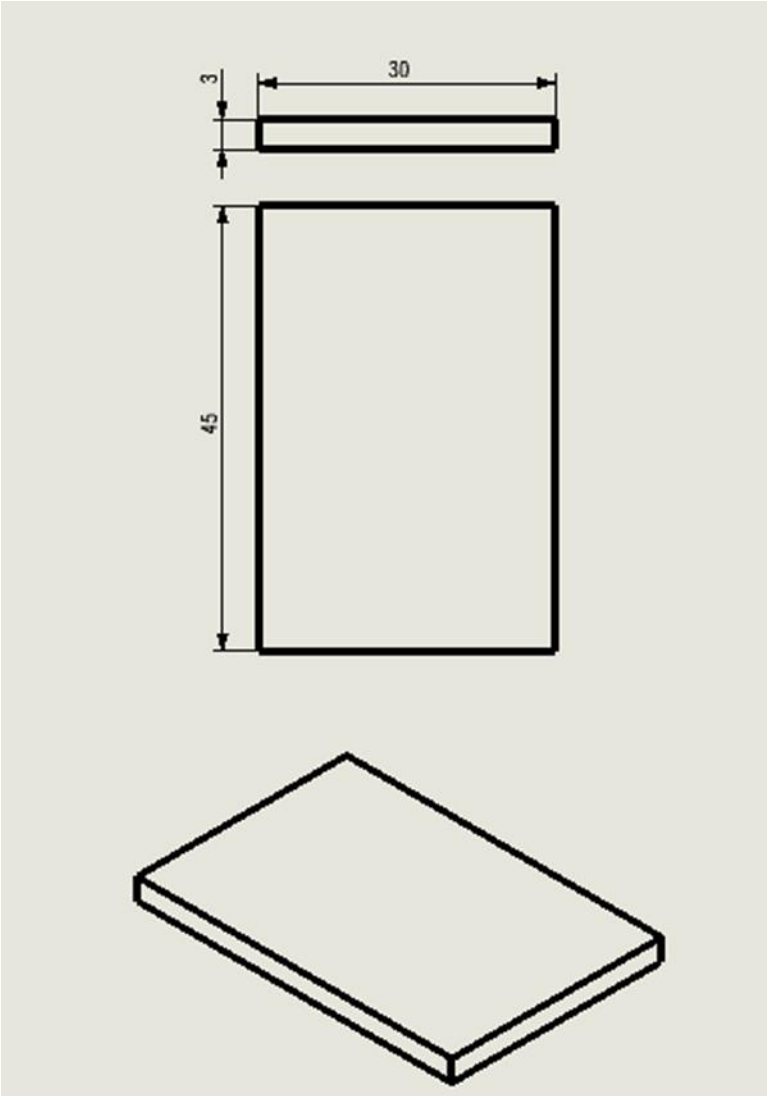


Imagen 19-Plano de unión de poleas

La imagen 11 muestra un diseño de Wall-H, la estructura principal esta conformada por dos planchuelas de aluminio de 6mm de espesor, unidas mediante una planchuela en la parte inferior para asi darle rigidez estructural, la parte superior esta sujeta por dos varillas roscadas de 8mm, desta manera ajustamos las dos planchuelas laterales de manera que queden paralelas, la transimicion esta pensada de manera simetrica, para que el robot se pueda mover tanto para adelante como para atrás sin tener problemas con los obstaculos, las orugas estan conformadas por una correa sinconica de paso L y grosor de 20 mm, ambas orugas deben mantener tensiones lo mas similares posibles para que el robot avance derecho, para esto se hacen ranuras en la plancha de aluminio lateral y se ponen resortes que empujen el motor hacia abajo, manteniendo tensas las orugas. Las 4 poleas inferiores de las esquinas estan sujetas robot mediante amortiguadores de resorte, estos le dan un mejor agarre al robot sobre terrenos irregulares ademas de evitar los golpes bruscos contra el piso. Luego a la estructura se adozan los distintos soportes para los sensores. la parte superior esta conformada por una estructura de placas de polipropileno y cuatro columnas de alumino en la cual se encuentran todas las placas y las dos baterias, también tiene una columna de alumnio que tiene en su extremo superior un motor paso a paso y aderido a este un caño de PVC termofusión.

2.1 Motores:

2.1.1 Descripción general:

Son los encargados de mover el robot los cuales lo moveran normalmente a 0,13m/s. La maxima veloicdad puede ser 0,15m/s. Se utilizan los motores MR08B-012004-44 de IGNIS.

2.1.2 Características técnicas:

Item	Valor	Unidad
Potencia	0,0067	Hp
Tensión nominal	12	V
Io. Inom. Is	0,20 . 0,99 . 7,92	A
Ruido máximo	100	Db
RPM Nom . RPM Vacío	6140 . 7600	RPM
Peso	0,160	Kg
Largo. Adicional por etapa. Diámetro Motor	107,7 . 5,5 . 27,5	Mm

Tabla 3- Características de los motores

Relación	Engranajes	Velocidad (RPM)	Cupla (Kgf.cm)	Cupla (Kgf.cm)	Cupla (Kgf.cm)	Cupla (Kgf.cm)
144.1	**664**	**44**	**11,07**	**14,83**	**38,49**	**38,49**

Tabla 4- Características de los motores

3.0 Módulo controlador de motores

3.0.1 Descripción general del módulo:

El proposito de este modulo es controlar el sentido de giro de los motores y la potencia que se le entrga a cada uno, los datos de velocidad son recibidos mediante I2C desde el sitema de control, en funcion de las velocidades recibidas (velocidad lineal y velocidad angular), se calcula la velocidad para cada motor, esta velocidad es un valor de 0 a 255 y con este valor controlamos la potencia en cada motor mediante la modulacion por ancho de pulso, este es un sistema de feedback, que obtiene los valores de velocidad lineal y de giro real mediante la utilizacion de encoders, los datos de corriente y velocidad lineal y angular son enviados nuevamente al sistema de control. Este modulo tambien se encarga del control de traccion.

3.0.2 Características técnicas:

CONTROLADOR:

Item	Valor	Unidad
Dimensiones	1	Cm
Microprocesador	ESP32	
Numero de motores a manejar	2	
Voltaje entrada tipico control	3.3-5	V
Temperatura maxima de operación	80	°C

Tabla 5- Caracteristicas del módulo controlador de los motores

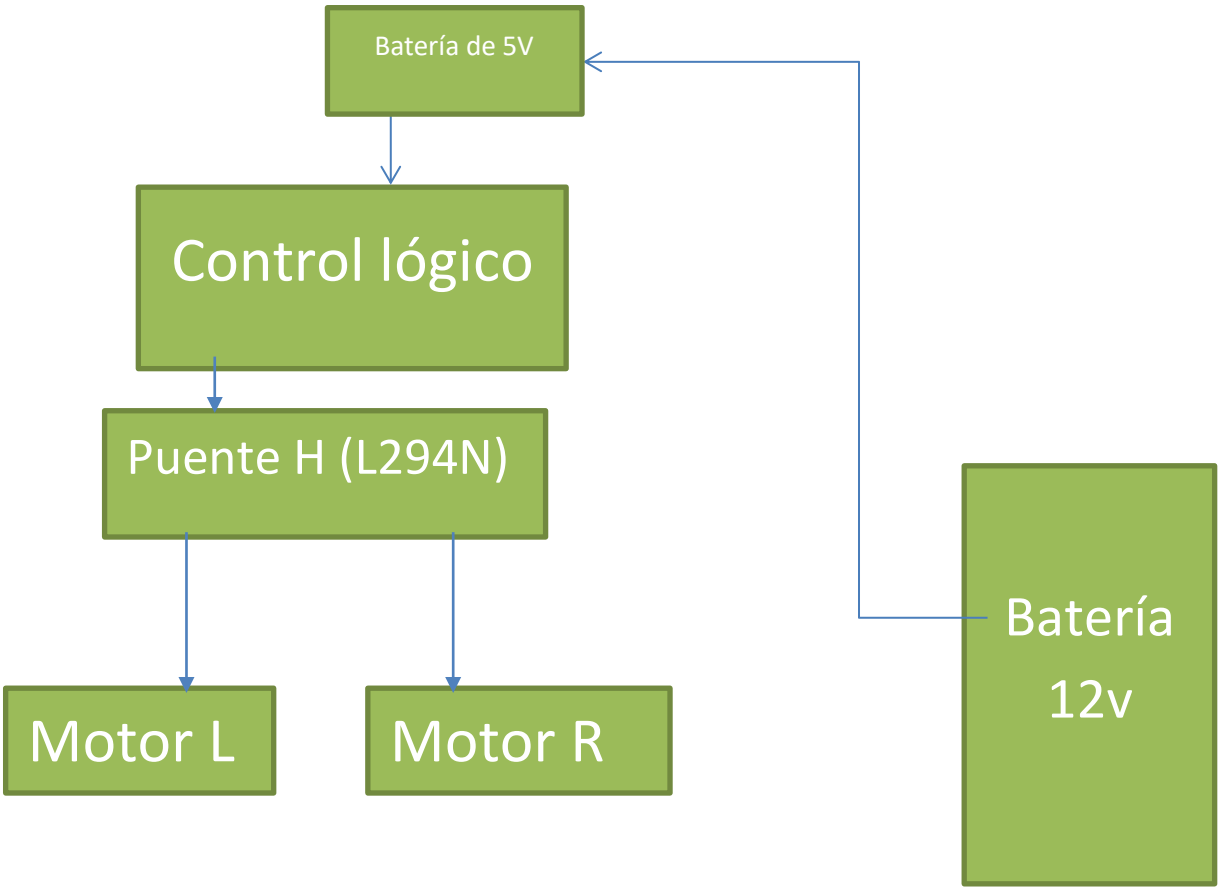
PUENTE H:

Item	Valor	Unidad
Dimensiones	5X5	Cm
Microprocesador	ESP32	
Numero de motores a manejar	2	
Voltaje entrada tipico control	5-12	V
Temperatura maxima de operación	80	°C
Voltaje de entrada tipico potencia	12	V

Tabla 6- Caracteristicas del puente H

3.0.3 Diagrama en bloques del sistema de control de motores:

Diagrama en bloques 1- Control de motores



3.1 Control lógico:

3.1.1 Descripción del control lógico:

El control lógico del módulo esta implementado a través de un microcontrolador ESP32, este recibe órdenes del sistema de control, estas órdenes son la dirección y velocidad, con estos datos se calcula la velocidad y sentido de giro de cada motor, mediante dos salidas de PWM para desplazar al robot de acuerdo con los datos recibidos.

3.1.2 Batería de 5V:

Se utiliza una batería de Iohn-litio de 5V con su propio controlador y regulador de carga

3.1.3 Control de energía:

Cuenta con dos indicares de nivel de batería, que nos muestran cuanta batería tiene el robot

4.0 Sistema de Control del robot:

4.0.1 Descripción y funciones del modulo ultrasónico:

Este modulo tiene como finalidad conocer la distancia de los obstaculos respectivamente del robot. Los sensores ultrasonicos se ubican estrategicamente para que este pueda identificar la posicion de los objetos y la proximidad con estos, la informacion de estos sensores sera procesada por el sistema de control para identificar objetos y poder esquivarlos la informacion sera leida de este modulo mediante el protocolo i2c por el sistema de control.

4.0.2 Diagrama en bloques módulo ultrasónico:

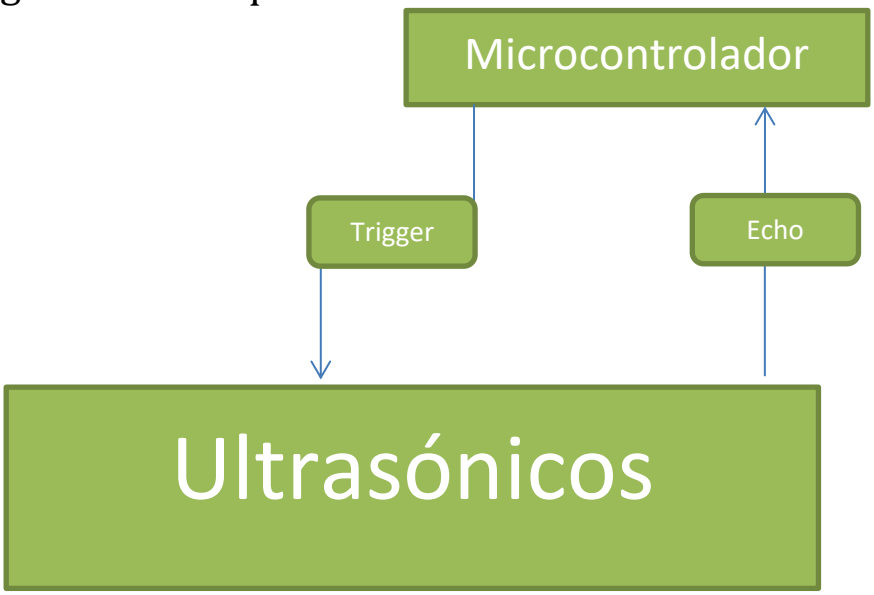


Diagrama en bloques 2- Ultrasónico

4.1 Sensores ultrasónicos:

4.1.1 Funcionamiento:

La deteccion de los obstaculos se logra mediante la emision de sonido y la deteccion del eco del mismo, si medimos el tiempo de vuelta del objeto entonces podremos saber la distancia a la que se encuentra el objeto.
En el caso de los Hc-sr04, debemos enviar un pulso de 20useg al terminal de trigger del sensor, y medir el tiempo en alto del pulso devuelto por el terminal echo del sensor, este tiempo equivale al tiempo transcurrido entre la emision y recepsion del sonido.

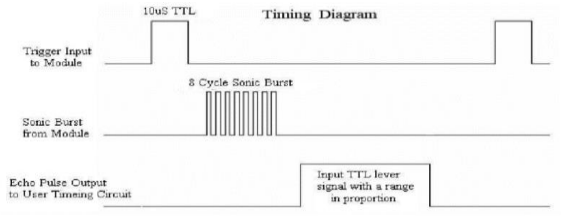


Imagen 21- Diagrama de tiempo del ultrasonico

4.1.2 Medición:

Ya que el robot posee 3 sensors ultrasonicos son controladas por el microcontrolador ESP32, el terminal echo se conecta al PIN 27, ya que mediante este se dispara la interrupcion externa en la ESP32, en este caso:

PIN27: ECHO

PIN 26: trigger

La medicion de tiempo en alto del pulso devuelto por el terminal echo se realiza mediante el timer1, este se pone en 0 al sensar el flanco ascendente y se toma el valor al sensar el flanco descendente, sabiendo la frecuencia del timer, el cual tiene un preescaler de 8, se puede calcular el timepo transcurrido y haciendo los respectivos calculos se sabe la distancia a la que se encuentra el obstaculo.

4.1.3 Programación:

```
import machine, time
from machine import Pin
import hcsr04


sensor = hcsr04.HCSR04(trigger_pin=13, echo_pin=12)
#sensor1 = hcsr04.HCSR04(trigger_pin=26, echo_pin=27)


while True:

  distance = sensor.distance_cm()
  distance = round(distance)

  print('Distance:', distance  , 'cm')
  time.sleep(1)
```

Programación 1- Sensor ultrasónico

4.1.4 Esquematico:

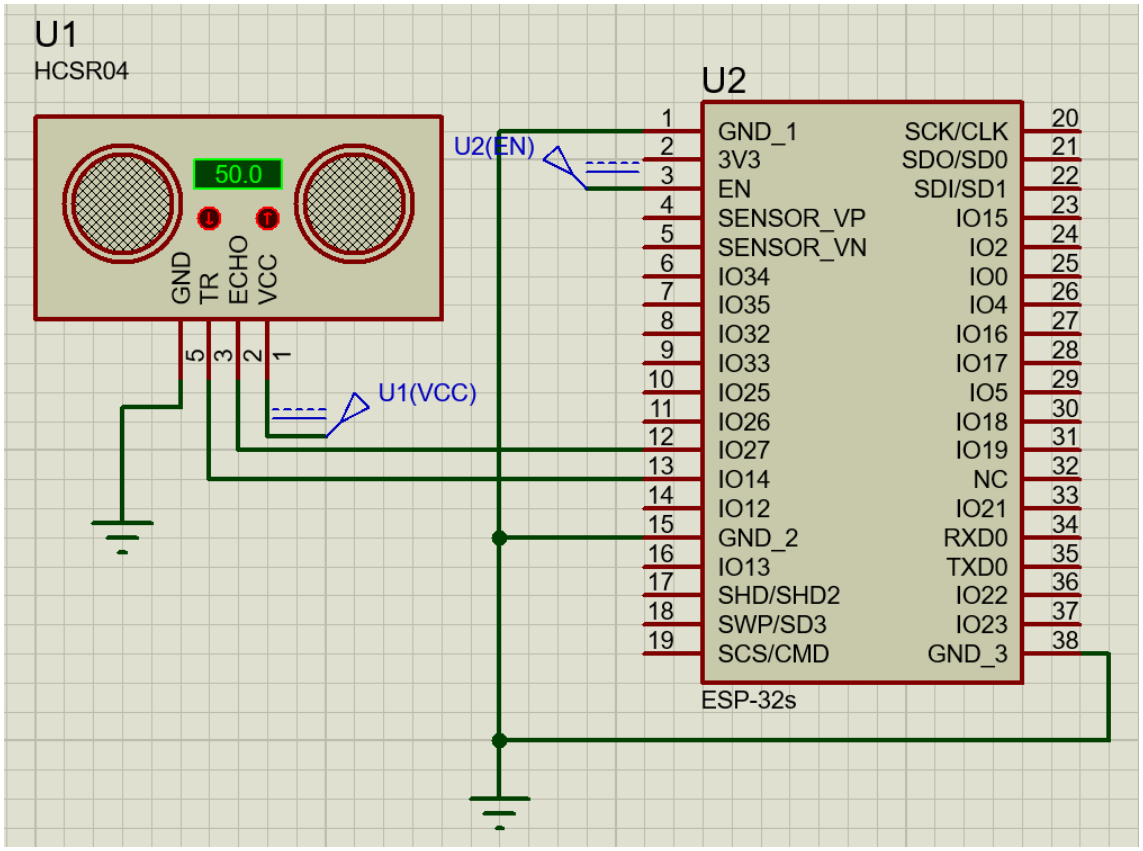


Imagen 20- Esquema del sensor ultrasonico

4.2.0 Módulo de cámara:

4.2.1 Descripción:

ESP32 CAM Modulo WiFi con Bluetooth y Camara OV2640 2MP, es una tarjeta de desarrollo que integra una pequeña cámara que puede funcionar de manera independiente.

La camara OV2640 de 2MP integra un sensor de imagen CMOS UXGA (1632*1232) de 1/4 de pulgada. El pequeño tamaño del sensor y el bajo voltaje de operación brindan todas las características de una sola cámara UXGA y un procesador de imágenes. A través del control de bus SCCB, puede generar datos de imagen de 8/10 bits de varias resoluciones, como fotograma completo, submuestreo, zoom y ventanas.

Imagen 22- Modulo Esp32 Cam Camara Wifi Bluetooth Ov2640 2mp Arduino

4.2.2 Utilización:

Utilizamos la cámara que viene con la ESP32 para poder mostrar por donde está el robot ya que la persona que lo meneje debe sabar por donde se encuentra el robot

4.2.3 Programación:

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

#include "fb_gfx.h"
#include "fd_forward.h"
#include "fr_forward.h"

#define ENROLL_CONFIRM_TIMES 5
#define FACE_ID_SAVE_NUMBER 7

#define FACE_COLOR_WHITE  0x00FFFFFF
#define FACE_COLOR_BLACK  0x00000000
#define FACE_COLOR_RED    0x000000FF
#define FACE_COLOR_GREEN  0x0000FF00
#define FACE_COLOR_BLUE   0x00FF0000
#define FACE_COLOR_YELLOW (FACE_COLOR_RED | FACE_COLOR_GREEN)
#define FACE_COLOR_CYAN   (FACE_COLOR_BLUE | FACE_COLOR_GREEN)
#define FACE_COLOR_PURPLE (FACE_COLOR_BLUE | FACE_COLOR_RED)

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;

#define PART_BOUNDARY "123456789000000000000987654321"
static const char* _STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;boundary=" PART_BOUNDARY;
static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
static const char* _STREAM_PART = "Content-Type: image/jpeg\r\nContent-Length: %u\r\n\r\n";

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

```
httpd_handle_t camera_httpd = NULL;

static mtmn_config_t mtmn_config = {0};
static int8_t detection_enabled = 0;
static int8_t recognition_enabled = 0;
static int8_t is_enrolling = 0;
static face_id_list id_list = {0};

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 void rgb_print(dl_matrix3du_t *image_matrix, uint32_t color, const char * str){
    fb_data_t fb;
    fb.width = image_matrix->w;
    fb.height = image_matrix->h;
    fb.data = image_matrix->item;
    fb.bytes_per_pixel = 3;
    fb.format = FB_BGR888;
    fb_gfx_print(&fb, (fb.width - (strlen(str) * 14)) / 2, 10, color, str);
}

static int rgb_printf(dl_matrix3du_t *image_matrix, uint32_t color, const char *format, ...){
    char loc_buf[64];
    char * temp = loc_buf;
    int len;
    va_list arg;
```

```
    va_list copy;
    va_start(arg, format);
    va_copy(copy, arg);
    len = vsnprintf(loc_buf, sizeof(loc_buf), format, arg);
    va_end(copy);
    if(len >= sizeof(loc_buf)){
        temp = (char*)malloc(len+1);
        if(temp == NULL) {
            return 0;
        }
    }
    vsnprintf(temp, len+1, format, arg);
    va_end(arg);
    rgb_print(image_matrix, color, temp);
    if(len > 64){
        free(temp);
    }
    return len;
}

static void draw_face_boxes(dl_matrix3du_t *image_matrix, box_array_t *boxes, int face_id){
    int x, y, w, h, i;
    uint32_t color = FACE_COLOR_YELLOW;
    if(face_id < 0){
        color = FACE_COLOR_RED;
    } else if(face_id > 0){
        color = FACE_COLOR_GREEN;
    }
    fb_data_t fb;
    fb.width = image_matrix->w;
    fb.height = image_matrix->h;
    fb.data = image_matrix->item;
    fb.bytes_per_pixel = 3;
    fb.format = FB_BGR888;
    for (i = 0; i < boxes->len; i++){
        // rectangle box
        x = (int)boxes->box[i].box_p[0];
        y = (int)boxes->box[i].box_p[1];
        w = (int)boxes->box[i].box_p[2] - x + 1;
        h = (int)boxes->box[i].box_p[3] - y + 1;
        fb_gfx_drawFastHLine(&fb, x, y, w, color);
        fb_gfx_drawFastHLine(&fb, x, y+h-1, w, color);
        fb_gfx_drawFastVLine(&fb, x, y, h, color);
        fb_gfx_drawFastVLine(&fb, x+w-1, y, h, color);
#if 0
        // landmark
        int x0, y0, j;
        for (j = 0; j < 10; j+=2) {
            x0 = (int)boxes->landmark[i].landmark_p[j];
            y0 = (int)boxes->landmark[i].landmark_p[j+1];
            fb_gfx_fillRect(&fb, x0, y0, 3, 3, color);
```

```
        }
#endif
    }
}

static int run_face_recognition(dl_matrix3du_t *image_matrix, box_array_t *net_boxes){
    dl_matrix3du_t *aligned_face = NULL;
    int matched_id = 0;

    aligned_face = dl_matrix3du_alloc(1, FACE_WIDTH, FACE_HEIGHT, 3);
    if(!aligned_face){
        Serial.println("Could not allocate face recognition buffer");
        return matched_id;
    }
    if (align_face(net_boxes, image_matrix, aligned_face) == ESP_OK){
        if (is_enrolling == 1){
            int8_t left_sample_face = enroll_face(&id_list, aligned_face);

            if(left_sample_face == (ENROLL_CONFIRM_TIMES - 1)){
                Serial.printf("Enrolling Face ID: %d\n", id_list.tail);
            }
            Serial.printf("Enrolling Face ID: %d sample %d\n", id_list.tail,
ENROLL_CONFIRM_TIMES - left_sample_face);
            rgb_printf(image_matrix, FACE_COLOR_CYAN, "ID[%u] Sample[%u]", id_list.tail,
ENROLL_CONFIRM_TIMES - left_sample_face);
            if (left_sample_face == 0){
                is_enrolling = 0;
                Serial.printf("Enrolled Face ID: %d\n", id_list.tail);
            }
        } else {
            matched_id = recognize_face(&id_list, aligned_face);
            if (matched_id >= 0) {
                Serial.printf("Match Face ID: %u\n", matched_id);
                rgb_printf(image_matrix, FACE_COLOR_GREEN, "Hello Subject %u", matched_id);
            } else {
                Serial.println("No Match Found");
                rgb_print(image_matrix, FACE_COLOR_RED, "Intruder Alert!");
                matched_id = -1;
            }
        }
    } else {
        Serial.println("Face Not Aligned");
        //rgb_print(image_matrix, FACE_COLOR_YELLOW, "Human Detected");
    }

    dl_matrix3du_free(aligned_face);
    return matched_id;
}

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.println("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");
    httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");

    size_t out_len, out_width, out_height;
    uint8_t * out_buf;
    bool s;
    bool detected = false;
    int face_id = 0;
    if(!detection_enabled || fb->width > 400){
        size_t fb_len = 0;
        if(fb->format == PIXFORMAT_JPEG){
            fb_len = fb->len;
            res = httpd_resp_send(req, (const char *)fb->buf, fb->len);
        } else {
            jpg_chunking_t jchunk = {req, 0};
            res = frame2jpg_cb(fb, 80, jpg_encode_stream, &jchunk)?ESP_OK:ESP_FAIL;
            httpd_resp_send_chunk(req, NULL, 0);
            fb_len = jchunk.len;
        }
        esp_camera_fb_return(fb);
        int64_t fr_end = esp_timer_get_time();
        Serial.printf("JPG: %uB %ums\n", (uint32_t)(fb_len), (uint32_t)((fr_end -
fr_start)/1000));
        return res;
    }

    dl_matrix3du_t *image_matrix = dl_matrix3du_alloc(1, fb->width, fb->height, 3);
    if (!image_matrix) {
```

```
        esp_camera_fb_return(fb);
        Serial.println("dl_matrix3du_alloc failed");
        httpd_resp_send_500(req);
        return ESP_FAIL;
    }

    out_buf = image_matrix->item;
    out_len = fb->width * fb->height * 3;
    out_width = fb->width;
    out_height = fb->height;

    s = fmt2rgb888(fb->buf, fb->len, fb->format, out_buf);
    esp_camera_fb_return(fb);
    if(!s){
        dl_matrix3du_free(image_matrix);
        Serial.println("to rgb888 failed");
        httpd_resp_send_500(req);
        return ESP_FAIL;
    }

    box_array_t *net_boxes = face_detect(image_matrix, &mtmn_config);

    if (net_boxes){
        detected = true;
        if(recognition_enabled){
            face_id = run_face_recognition(image_matrix, net_boxes);
        }
        draw_face_boxes(image_matrix, net_boxes, face_id);
        free(net_boxes->score);
        free(net_boxes->box);
        free(net_boxes->landmark);
        free(net_boxes);
    }

    jpg_chunking_t jchunk = {req, 0};
    s = fmt2jpg_cb(out_buf, out_len, out_width, out_height, PIXFORMAT_RGB888, 90,
jpg_encode_stream, &jchunk);
    dl_matrix3du_free(image_matrix);
    if(!s){
        Serial.println("JPEG compression failed");
        return ESP_FAIL;
    }

    int64_t fr_end = esp_timer_get_time();
    Serial.printf("FACE: %uB %ums %s%d\n", (uint32_t)(jchunk.len), (uint32_t)((fr_end -
fr_start)/1000), detected?"DETECTED ":"", face_id);
    return res;
}

static esp_err_t stream_handler(httpd_req_t *req){
    camera_fb_t * fb = NULL;
```

```
esp_err_t res = ESP_OK;
size_t _jpg_buf_len = 0;
uint8_t * _jpg_buf = NULL;
char * part_buf[64];
dl_matrix3du_t *image_matrix = NULL;
bool detected = false;
int face_id = 0;
int64_t fr_start = 0;
int64_t fr_ready = 0;
int64_t fr_face = 0;
int64_t fr_recognize = 0;
int64_t fr_encode = 0;

static int64_t last_frame = 0;
if(!last_frame) {
    last_frame = esp_timer_get_time();
}

res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
if(res != ESP_OK){
    return res;
}

httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");

while(true){
    detected = false;
    face_id = 0;
    fb = esp_camera_fb_get();
    if (!fb) {
        Serial.println("Camera capture failed");
        res = ESP_FAIL;
    } else {
        fr_start = esp_timer_get_time();
        fr_ready = fr_start;
        fr_face = fr_start;
        fr_encode = fr_start;
        fr_recognize = fr_start;
        if(!detection_enabled || fb->width > 400){
            if(fb->format != PIXFORMAT_JPEG){
                bool jpeg_converted = frame2jpg(fb, 80, &_jpg_buf, &_jpg_buf_len);
                esp_camera_fb_return(fb);
                fb = NULL;
                if(!jpeg_converted){
                    Serial.println("JPEG compression failed");
                    res = ESP_FAIL;
                }
            } else {
                _jpg_buf_len = fb->len;
                _jpg_buf = fb->buf;
            }
```

```
            } else {

                image_matrix = dl_matrix3du_alloc(1, fb->width, fb->height, 3);

                if (!image_matrix) {
                    Serial.println("dl_matrix3du_alloc failed");
                    res = ESP_FAIL;
                } else {
                    if(!fmt2rgb888(fb->buf, fb->len, fb->format, image_matrix->item)){
                        Serial.println("fmt2rgb888 failed");
                        res = ESP_FAIL;
                    } else {
                        fr_ready = esp_timer_get_time();
                        box_array_t *net_boxes = NULL;
                        if(detection_enabled){
                            net_boxes = face_detect(image_matrix, &mtmn_config);
                        }
                        fr_face = esp_timer_get_time();
                        fr_recognize = fr_face;
                        if (net_boxes || fb->format != PIXFORMAT_JPEG){
                            if(net_boxes){
                                detected = true;
                                if(recognition_enabled){
                                    face_id = run_face_recognition(image_matrix, net_boxes);
                                }
                                fr_recognize = esp_timer_get_time();
                                draw_face_boxes(image_matrix, net_boxes, face_id);
                                free(net_boxes->score);
                                free(net_boxes->box);
                                free(net_boxes->landmark);
                                free(net_boxes);
                            }
                            if(!fmt2jpg(image_matrix->item, fb->width*fb->height*3, fb->width,
fb->height, PIXFORMAT_RGB888, 90, &_jpg_buf, &_jpg_buf_len)){
                                Serial.println("fmt2jpg failed");
                                res = ESP_FAIL;
                            }
                            esp_camera_fb_return(fb);
                            fb = NULL;
                        } else {
                            _jpg_buf = fb->buf;
                            _jpg_buf_len = fb->len;
                        }
                        fr_encode = esp_timer_get_time();
                    }
                    dl_matrix3du_free(image_matrix);
                }
            }
        }
        if(res == ESP_OK){
            res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY, strlen(_STREAM_BOUNDARY));
```

```
        }
        if(res == ESP_OK){
            size_t hlen = snprintf((char *)part_buf, 64, _STREAM_PART, _jpg_buf_len);
            res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
        }
        if(res == ESP_OK){
            res = httpd_resp_send_chunk(req, (const char *)_jpg_buf, _jpg_buf_len);
        }
        if(fb){
            esp_camera_fb_return(fb);
            fb = NULL;
            _jpg_buf = NULL;
        } else if(_jpg_buf){
            free(_jpg_buf);
            _jpg_buf = NULL;
        }
        if(res != ESP_OK){
            break;
        }
        int64_t fr_end = esp_timer_get_time();

        int64_t ready_time = (fr_ready - fr_start)/1000;
        int64_t face_time = (fr_face - fr_ready)/1000;
        int64_t recognize_time = (fr_recognize - fr_face)/1000;
        int64_t encode_time = (fr_encode - fr_recognize)/1000;
        int64_t process_time = (fr_encode - fr_start)/1000;

        int64_t frame_time = fr_end - last_frame;
        last_frame = fr_end;
        frame_time /= 1000;
        uint32_t avg_frame_time = ra_filter_run(&ra_filter, frame_time);
        Serial.printf("MJPG: %uB %ums (%.1ffps), AVG: %ums (%.1ffps), %u+%u+%u+%u=%u %s%d\n",
            (uint32_t)(_jpg_buf_len),
            (uint32_t)frame_time, 1000.0 / (uint32_t)frame_time,
            avg_frame_time, 1000.0 / avg_frame_time,
            (uint32_t)ready_time, (uint32_t)face_time, (uint32_t)recognize_time,
(uint32_t)encode_time, (uint32_t)process_time,
            (detected)?"DETECTED ":"", face_id
        );
    }

    last_frame = 0;
    return res;
}

static esp_err_t cmd_handler(httpd_req_t *req){
    char*  buf;
    size_t buf_len;
    char variable[32] = {0,};
    char value[32] = {0,};
```

```
    buf_len = httpd_req_get_url_query_len(req) + 1;
    if (buf_len > 1) {
        buf = (char*)malloc(buf_len);
        if(!buf){
            httpd_resp_send_500(req);
            return ESP_FAIL;
        }
        if (httpd_req_get_url_query_str(req, buf, buf_len) == ESP_OK) {
            if (httpd_query_key_value(buf, "var", variable, sizeof(variable)) == ESP_OK &&
                httpd_query_key_value(buf, "val", value, sizeof(value)) == ESP_OK) {
            } else {
                free(buf);
                httpd_resp_send_404(req);
                return ESP_FAIL;
            }
        } else {
            free(buf);
            httpd_resp_send_404(req);
            return ESP_FAIL;
        }
        free(buf);
    } else {
        httpd_resp_send_404(req);
        return ESP_FAIL;
    }

    int val = atoi(value);
    sensor_t * s = esp_camera_sensor_get();
    int res = 0;

    if(!strcmp(variable, "framesize")) {
        if(s->pixformat == PIXFORMAT_JPEG) res = s->set_framesize(s, (framesize_t)val);
    }
    else if(!strcmp(variable, "quality")) res = s->set_quality(s, val);
    else if(!strcmp(variable, "contrast")) res = s->set_contrast(s, val);
    else if(!strcmp(variable, "brightness")) res = s->set_brightness(s, val);
    else if(!strcmp(variable, "saturation")) res = s->set_saturation(s, val);
    else if(!strcmp(variable, "gainceiling")) res = s->set_gainceiling(s, (gainceiling_t)val);
    else if(!strcmp(variable, "colorbar")) res = s->set_colorbar(s, val);
    else if(!strcmp(variable, "awb")) res = s->set_whitebal(s, val);
    else if(!strcmp(variable, "agc")) res = s->set_gain_ctrl(s, val);
    else if(!strcmp(variable, "aec")) res = s->set_exposure_ctrl(s, val);
    else if(!strcmp(variable, "hmirror")) res = s->set_hmirror(s, val);
    else if(!strcmp(variable, "vflip")) res = s->set_vflip(s, val);
    else if(!strcmp(variable, "awb_gain")) res = s->set_awb_gain(s, val);
    else if(!strcmp(variable, "agc_gain")) res = s->set_agc_gain(s, val);
    else if(!strcmp(variable, "aec_value")) res = s->set_aec_value(s, val);
    else if(!strcmp(variable, "aec2")) res = s->set_aec2(s, val);
    else if(!strcmp(variable, "dcw")) res = s->set_dcw(s, val);
    else if(!strcmp(variable, "bpc")) res = s->set_bpc(s, val);
    else if(!strcmp(variable, "wpc")) res = s->set_wpc(s, val);
```

```
    else if(!strcmp(variable, "raw_gma")) res = s->set_raw_gma(s, val);
    else if(!strcmp(variable, "lenc")) res = s->set_lenc(s, val);
    else if(!strcmp(variable, "special_effect")) res = s->set_special_effect(s, val);
    else if(!strcmp(variable, "wb_mode")) res = s->set_wb_mode(s, val);
    else if(!strcmp(variable, "ae_level")) res = s->set_ae_level(s, val);
    else if(!strcmp(variable, "face_detect")) {
        detection_enabled = val;
        if(!detection_enabled) {
            recognition_enabled = 0;
        }
    }
    else if(!strcmp(variable, "face_enroll")) is_enrolling = val;
    else if(!strcmp(variable, "face_recognize")) {
        recognition_enabled = val;
        if(recognition_enabled){
            detection_enabled = val;
        }
    }
    else {
        res = -1;
    }

    if(res){
        return httpd_resp_send_500(req);
    }

    httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
    return httpd_resp_send(req, NULL, 0);
}

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++ = '{';

    p+=sprintf(p, "\"framesize\":%u,", s->status.framesize);
    p+=sprintf(p, "\"quality\":%u,", s->status.quality);
    p+=sprintf(p, "\"brightness\":%d,", s->status.brightness);
    p+=sprintf(p, "\"contrast\":%d,", s->status.contrast);
    p+=sprintf(p, "\"saturation\":%d,", s->status.saturation);
    p+=sprintf(p, "\"sharpness\":%d,", s->status.sharpness);
    p+=sprintf(p, "\"special_effect\":%u,", s->status.special_effect);
    p+=sprintf(p, "\"wb_mode\":%u,", s->status.wb_mode);
    p+=sprintf(p, "\"awb\":%u,", s->status.awb);
    p+=sprintf(p, "\"awb_gain\":%u,", s->status.awb_gain);
    p+=sprintf(p, "\"aec\":%u,", s->status.aec);
    p+=sprintf(p, "\"aec2\":%u,", s->status.aec2);
    p+=sprintf(p, "\"ae_level\":%d,", s->status.ae_level);
    p+=sprintf(p, "\"aec_value\":%u,", s->status.aec_value);
```

```
    p+=sprintf(p, "\"agc\":%u,", s->status.agc);
    p+=sprintf(p, "\"agc_gain\":%u,", s->status.agc_gain);
    p+=sprintf(p, "\"gainceiling\":%u,", s->status.gainceiling);
    p+=sprintf(p, "\"bpc\":%u,", s->status.bpc);
    p+=sprintf(p, "\"wpc\":%u,", s->status.wpc);
    p+=sprintf(p, "\"raw_gma\":%u,", s->status.raw_gma);
    p+=sprintf(p, "\"lenc\":%u,", s->status.lenc);
    p+=sprintf(p, "\"vflip\":%u,", s->status.vflip);
    p+=sprintf(p, "\"hmirror\":%u,", s->status.hmirror);
    p+=sprintf(p, "\"dcw\":%u,", s->status.dcw);
    p+=sprintf(p, "\"colorbar\":%u,", s->status.colorbar);
    p+=sprintf(p, "\"face_detect\":%u,", detection_enabled);
    p+=sprintf(p, "\"face_enroll\":%u,", is_enrolling);
    p+=sprintf(p, "\"face_recognize\":%u", recognition_enabled);
    *p++ = '}';
    *p++ = 0;
    httpd_resp_set_type(req, "application/json");
    httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
    return httpd_resp_send(req, json_response, strlen(json_response));
}

static esp_err_t index_handler(httpd_req_t *req){
    httpd_resp_set_type(req, "text/html");
    httpd_resp_set_hdr(req, "Content-Encoding", "gzip");
    sensor_t * s = esp_camera_sensor_get();
    if (s->id.PID == OV3660_PID) {
        return httpd_resp_send(req, (const char *)index_ov3660_html_gz,
index_ov3660_html_gz_len);
    }
    return httpd_resp_send(req, (const char *)index_ov2640_html_gz, index_ov2640_html_gz_len);
}

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

    httpd_uri_t index_uri = {
        .uri       = "/",
        .method    = HTTP_GET,
        .handler   = index_handler,
        .user_ctx  = NULL
    };

    httpd_uri_t status_uri = {
        .uri       = "/status",
        .method    = HTTP_GET,
        .handler   = status_handler,
        .user_ctx  = NULL
    };

    httpd_uri_t cmd_uri = {
        .uri       = "/control",
```

```
        .method    = HTTP_GET,
        .handler   = cmd_handler,
        .user_ctx  = NULL
    };

    httpd_uri_t capture_uri = {
        .uri       = "/capture",
        .method    = HTTP_GET,
        .handler   = capture_handler,
        .user_ctx  = NULL
    };

   httpd_uri_t stream_uri = {
        .uri       = "/stream",
        .method    = HTTP_GET,
        .handler   = stream_handler,
        .user_ctx  = NULL
    };


    ra_filter_init(&ra_filter, 20);

    mtmn_config.type = FAST;
    mtmn_config.min_face = 80;
    mtmn_config.pyramid = 0.707;
    mtmn_config.pyramid_times = 4;
    mtmn_config.p_threshold.score = 0.6;
    mtmn_config.p_threshold.nms = 0.7;
    mtmn_config.p_threshold.candidate_number = 20;
    mtmn_config.r_threshold.score = 0.7;
    mtmn_config.r_threshold.nms = 0.7;
    mtmn_config.r_threshold.candidate_number = 10;
    mtmn_config.o_threshold.score = 0.7;
    mtmn_config.o_threshold.nms = 0.7;
    mtmn_config.o_threshold.candidate_number = 1;

    face_id_init(&id_list, FACE_ID_SAVE_NUMBER, ENROLL_CONFIRM_TIMES);

    Serial.printf("Starting web server on port: '%d'\n", config.server_port);
    if (httpd_start(&camera_httpd, &config) == ESP_OK) {
        httpd_register_uri_handler(camera_httpd, &index_uri);
        httpd_register_uri_handler(camera_httpd, &cmd_uri);
        httpd_register_uri_handler(camera_httpd, &status_uri);
        httpd_register_uri_handler(camera_httpd, &capture_uri);
    }

    config.server_port += 1;
    config.ctrl_port += 1;
    Serial.printf("Starting stream server on port: '%d'\n", config.server_port);
    if (httpd_start(&stream_httpd, &config) == ESP_OK) {
        httpd_register_uri_handler(stream_httpd, &stream_uri);
```

```
    }
}
```

Programación 2- Cámara

4.3.0 Joystick de control:

4.3.1 Descripción:

Al igual que un joystick de una consola de juegos puede contolar los ejes X e Y con este modulo. Hay que conectar los terminales VRx y VRy a pines analógicos y obtenga la posición de la palanca. Conecte el terminal SW a una entrada digital y sense si se está presionando la palanca hacia abajo.

4.3.2 Utilización:

Usamos el joystick para poder controlar el movimiento del robot que va a tener en el lugar que se maneje.

4.3.3 Programación:

```
from machine import ADC,Pin
import time, usocket, network, _thread

adc0=ADC(Pin(36))               #create ADC object
adc1=ADC(Pin(39))
adc2=ADC(Pin(34))
dato = ""
adc1.atten(ADC.ATTN_11DB)
adc2.atten(ADC.ATTN_11DB)
led=Pin(2,Pin.OUT)

def thread1():
  while True:
    led.value(1)            #Set led turn on
    time.sleep(0.5)
    led.value(0)            #Set led turn off
    time.sleep(0.5)

sta_if = network.WLAN(network.STA_IF)

logic_state2 = 1

if not sta_if.isconnected():
    print('connecting to network...')
    sta_if.active(True)
    sta_if.connect('esp','12345678')
    while not sta_if.isconnected():
      pass

sta_if.ifconfig(("192.168.100.207","255.255.255.0","192.168.100.2","8.8.8.8"))
print('network config:', sta_if.ifconfig()[0])
```

```
s = usocket.socket()
s.connect(("192.168.100.205",2020))
print("conectado")

_thread.start_new_thread(thread1,())

while True:
  vry = adc1.read()
  vrx = 4095 - adc2.read()
  sw = adc0.read()

  if vrx > 4050:
    print("GirarAtras()")
    dato = "GirarAtras()"
  elif vrx < 50:
    print("Adelante()")
    dato = "Adelante()"
  elif vry < 50:
    print("GirarIzq()")
    dato = "GirarIzq()"
  elif vry > 4050:
    print("GirarDer()")
    dato = "GirarDer()"
  elif sw < 50:
    print("Boton apretado")
    dato = "Boton()"
  else:
    print("nada apretado")
    dato = "Nada"
  print("")
  s.send(dato.encode())
  time.sleep(1.3)
```

Programación 3- Joystick

4.4 Control con el puente H:

Imagen 23- Puente H

4.4.1 Descrpción:

El Doble Puente H L298N es el modulo utilizado para controlar 2 motores o un motor paso a paso bipolar/unipolar. Es capaz de controlar corriente de hasta 2A.
Este modulo ademas te permite controlar el sentido y velocidad de giro de motores mediante señales TTL que se pueden obtener desde la ESP32

4.4.2 Utilización:

Utilizamos el puente H para poder controlar la alimentacion, el sentido y la potencia de los motores.

4.4.3 Programación:

```
from machine import Pin, I2C
import network, time, usocket, _thread
from libreria import HCSR04, Motor, SensorBase, MLX90614, max30100
import stepper

motores = Motor(16, Pin.OUT,17, Pin.OUT,18, Pin.OUT,19, Pin.OUT)
sta_if = network.WLAN(network.STA_IF)
led=Pin(2,Pin.OUT)
sensora1 = HCSR04(trigger_pin=27, echo_pin=26)
sensora2 = HCSR04(trigger_pin=12, echo_pin=10)
sensorb3 = HCSR04(trigger_pin=12, echo_pin=13)
i2c = I2C(scl=Pin(22), sda=Pin(21), freq=100000)
sensormlx = MLX90614(i2c)
sensormax = max30100.MAX30100(i2c=i2c)
sensormax.enable_spo2()
s1 = create(Pin(15,Pin.OUT), Pin(2,Pin.OUT), Pin(0,Pin.OUT), Pin(4,Pin.OUT), delay=2)
boton = Pin(23, Pin.IN)

def thread1():
  #Blink
  while True:
    led.value(1)            #Set led turn on
```

```
    time.sleep(0.5)
    led.value(0)            #Set led turn off
    time.sleep(0.5)

def thread2():
  #Control de motores
  (sc,addr) = s.accept()
  print(addr)
  continuar2=True
  while continuar2:
    mensaje = sc.recv(64)
    print(str(mensaje[0:12]),"primero")

    if str(mensaje[0:14]) == "b'Adelante()'":
      print("Adelante()")
      motores.Adelante()

    elif str(mensaje[0:14]) == "b'GirarIzq()'":
      print("GirarIzq()")
      motores.GirarIzq()

    elif str(mensaje[0:14]) == "b'GirarDer()'":
      print("GirarDer()")
      motores.GirarDer()

    elif str(mensaje[0:16]) == "b'GirarAtras()'":
      print("GirarAtras()")
      motores.Atras()

    elif str(mensaje[0:10]) == "b'Boton()'":
      if logic_state2 == 1:
        s1.angle(60)
        logic_state2 += 1
        print("la sube")
      elif logic_state2 == 2:
        s1.angle(60,-1)
        logic_state2 -= 1
        print("la baja")

    elif str(mensaje[0:7]) == "b'Nada'":
      print("Parar")
      motores.Parar()


    if distancea1 <= 30 and distancea1 != 0:
      motores.Parar()
      print("Sensor A1 detecto")

    elif distancea2 <= 30 and distancea2 != 0:
      motores.Parar()
      print("Sensor A2 detecto")
```

```
    elif distanceb3 <= 30 and distanceb3 != 0:
      motores.Parar()
      print("Sensor B3 detecto")

    time.sleep(0.6)
    sc.close()
  s.close()

def thread3():
  #Control de ultrasonicos
  distancea1 = round(sensora1.distance_cm())
  distancea2 = round(sensora2.distance_cm())
  distanceb3 = round(sensorb3.distance_cm())
  time.sleep(0.6)

def thread4():
  #control de sensor de temperatura
  while True:
    contador = 0
    momento = round(sensormlx.read_object_temp(),2) +3
    if sensormlx >= 30 and sensormlx <= 40:
      for i in range (30):
        temperaturas.append(round(sensormlx.read_object_temp(),2) +3)
        print(" Temperatura de la persona: ", round(sensormlx.read_object_temp(),2) +3)
        time.sleep(1)
      for i in temperaturas[15:30]:
        if contador == 0:
          contador = contador + 1
          pass
        else:
          temperaturas[0] = temperaturas[0] + i
      temperature = temperaturas[0] / 15

      if temperature > 38:
        print("Tenes fiebre y tu temperatura es ", temperature)
      else:
        print("No tenes fiebre y tu temperatura es ", temperature)
    else:
      temperaturas = []

def thread5():
  #Control sensor oximetro
  while True:
    sensormax.read_sensor()
    #print(sensor.ir, sensor.red)
    rawspo2 = sensormax.ir
    rawheartrate = sensormax.red

    spo2 = rawspo2/100
    heartrate = rawheartrate/200
```

```
    if spo2 > 100 :
      spo2 = 99.9
    elif spo2 < 93.0 and spo2 > 50.0 :
      spo2 = 93.0
    elif spo2 < 49 :
      spo2 = 0.0
    else :
      spo2 = spo2

    if heartrate > 130 :

      heartrate = 150
    elif heartrate < 65 and heartrate > 50 :
      heartrate = 65.0
    elif heartrate < 49 :
      heartrate = 0.0
    else :
      heartrate = heartrate

    #os.system("cls")
    if spo2 > 30:
      print("Oxigeno en sangre: ", spo2,"%         Ritmo Card閾哑co: ", heartrate)
    else:
      print("Tomando mediciones, porfavor espere.")




if not sta_if.isconnected():
    print('connecting to network...')
    sta_if.active(True)
    sta_if.connect('esp', '12345678')
    while not sta_if.isconnected():
        pass

sta_if.ifconfig(('192.168.100.205','255.255.255.0','192.168.100.2','8.8.8.8'))
print('network config:', sta_if.ifconfig()[0])

print("Conexion establecida")
s= usocket.socket()
s.bind((sta_if.ifconfig()[0],2020))
print("bindeado")
s.listen(10)

motores.Parar()
print("Server iniciado, esperando conexiones")
_thread.start_new_thread(thread1,())

logic_state2 = 1

_thread.start_new_thread(thread3,())
```

```
_thread.start_new_thread(thread2,())
_thread.start_new_thread(thread4,())
_thread.start_new_thread(thread5,())
```

Programación 4- Motores

5.0 Sistema de baterías:

5.0.1 Descripción:

El sistema de baterías se encarga de alimentar los distintos componentes del robot, este debe ser capaz de administrar la carga de las baterías, proteger a los circuitos del robot ante picos de consumo de corriente y mantener una alimentación constante durante el trabajo del Wall-H. Las tensiones que debe entregar son 12v y 5v.

6.0 Sistemas de comunicaciones:

6.0.1 Introducción:

Los sistemas de comunicaciones de Wall-H pueden dividirse un dos súper-sistemas: un sistema interno para la comunicación entre módulos y un sistema para la comunicación con equipos externos. El primero se basa en tecnologías de comunicación de corto alcance y fácil expansión; las del segundo, en tecnologías de medio y largo alcance con enlace punto a punto a través de una red Wifi.

Estos sistemas forman parte de la estructura esencial de Wall-H y por lo tanto no son removibles.

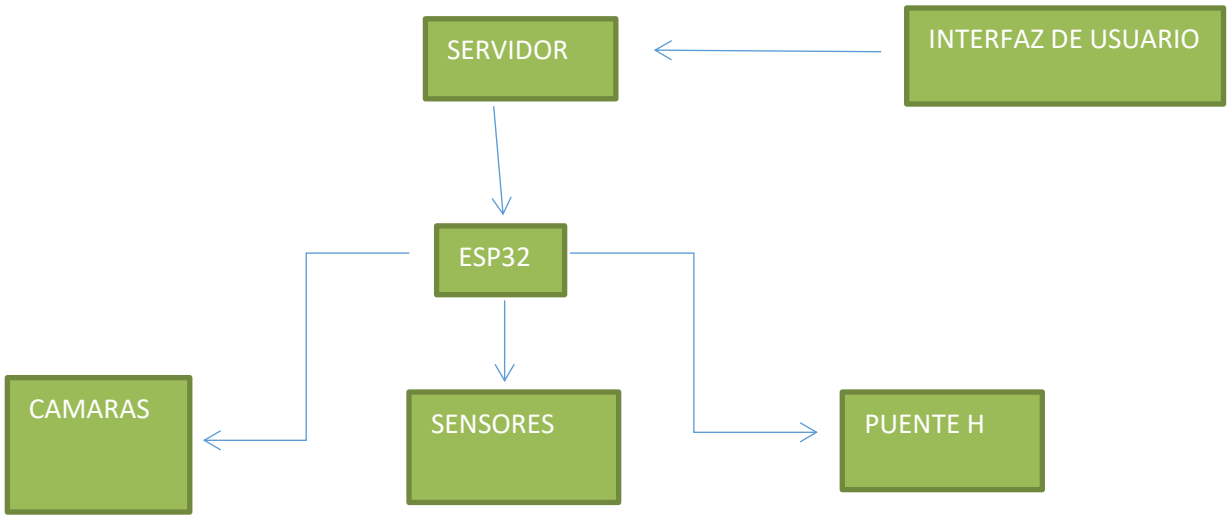


Diagrama en bloques 3- sistema de comunicaciones

6.1.0 Sistemas de comunicación internos:

Estos sistemas tienen una topología de bus, haciendo más sencillo el agregado de nuevos dispositivos. El bus I2C es el estándar para la comunicación entre los módulos.

6.1.1 Bus I2C:

El ESP32 tiene dos interfaces de bus I2C que pueden servir como maestro o esclavo I2C. I²C significa Inter Integrado Circuit (es pronunciado I-cuadrado-C), y es una, multi-master, protocolo de comunicación multi-esclavo síncrona.

Ejemplo:

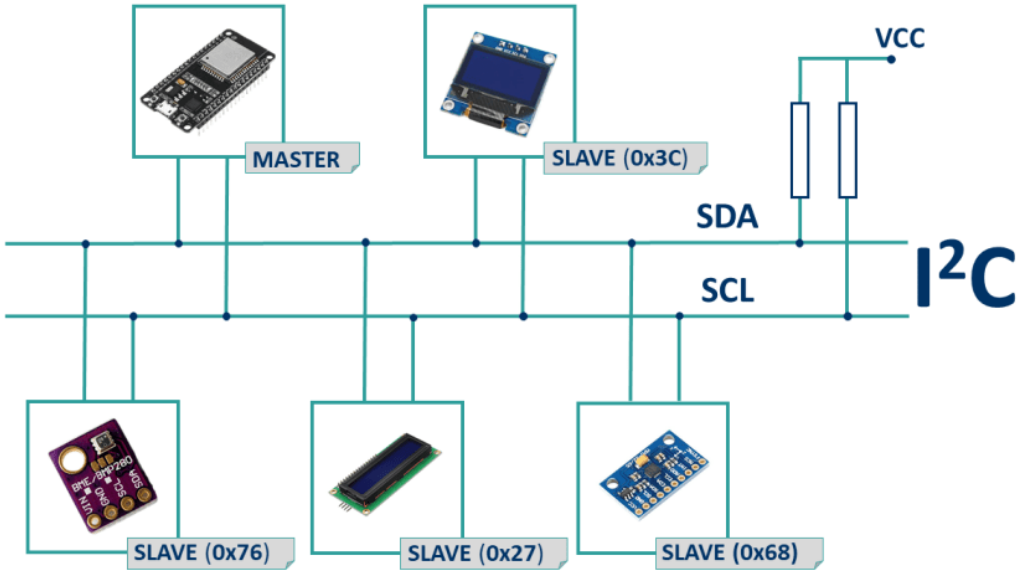


Imagen 24- Ejemplo del trabajo de I2C

I2C es un bus serie sincrónico de arquitectura master-slave. Es ampliamente utilizado como interfaz en sensores y microcontroladores.

El bus de I2C es compartido por todos los módulos y es accesible desde una bornera que une las líneas de transmisión con el puerto GPIO. Cada dispositivo se enlaza a través de dos líneas, una de datos y otra de clock. Las líneas ya poseen resistencias de pull-up, por lo que no son requeridas en los dispositivos.

I2C	I2CEXT0_SCL_in	Any GPIO	Two I2C devices in slave or master modes
	I2CEXT0_SDA_in		
	I2CEXT1_SCL_in		
	I2CEXT1_SDA_in		
	I2CEXT0_SCL_out		
	I2CEXT0_SDA_out		
	I2CEXT1_SCL_out		
	I2CEXT1_SDA_out		

Imagen 25- I2C en datasheet de ESP32

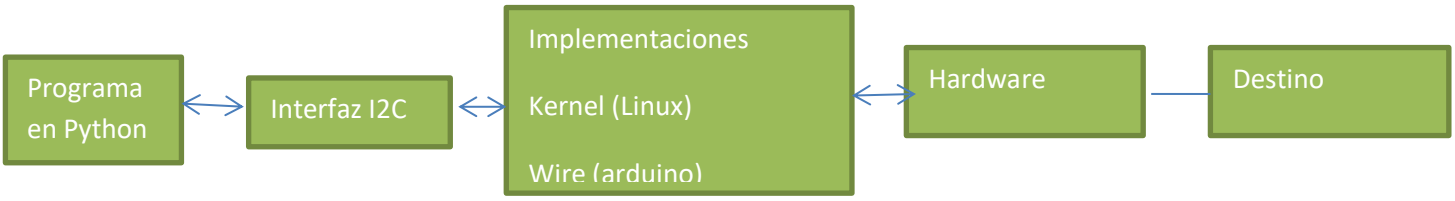


Diagrama en bloques 4- I2C

6.2 Sistema de comunicación con equipos externos:

Para la comunicación con sistemas de control externo se utilizan enlaces de Wifi. Estas tecnologías pueden proporcionar un alcance de 300m y más de 1km, respectivamente. Sin embargo, las antenas colocadas actualmente permiten un alcance efectivo de 100m y 50m en espacios abiertos.

6.2.1 Enlace Wifi:

El DPC escanea las redes Wifi disponibles a su alrededor y busca redes de una lista de AP (Access Point) conocidos. Si encuentra alguno, se conecta a él. En caso contrario, el DPC crea su propio AP. El DPC tiene DHCP activado, por lo que el AP determina su IP en la red; en el caso de estar en su red generada, se encuentra en la dirección 192.168.100.1

La mayoría de transacciones de datos sobre este enlace se realizan sobre el protocolo TCP.

6.3 Comunicación del servidor de control:

El sistema de control externo está compuesto un servidor que asocia su red ip local con el puerto 2020 y el cliente se conecta a la misma. Ambos programas están programados en Python para la plataforma .NET (compatible con sistemas Windows, Linux y MacOS).

6.3.1 Descubrimiento del servidor:

Al iniciar el cliente se inicia un proceso para descubrir la presencia de un servidor en la red y su dirección IP:

1. El cliente solicita la configuración de los adaptadores de red para descubrir su dirección IP local y la máscara de subred. Con eso calcula la dirección de broadcast.
2. El cliente envía un paquete UDP con la solicitud de descubrimiento (DSP), desde el puerto 13002 hacia el puerto 13001 de la dirección de broadcast.
3. El AP reenvía el paquete a todos los dispositivos conectados.
4. El servidor recibe el paquete en el puerto 13001 y, si la solicitud es válida, envía un paquete UDP al puerto 13003 de la dirección IP de origen del paquete (el cliente) con la respuesta a la solicitud (RSP).
5. El cliente recibe el paquete y, si la respuesta es válida, guarda la dirección IP de origen del paquete (el servidor) para luego comenzar la conexión TCP punto a punto.

6.3.2 Sistema de protocolos:

Los protocolos son funciones que determinan un intercambio de mensajes. Están compuestos por un nombre que los identifica y un delegado ProtocolDelegate que realiza el intercambio de mensajes.

Los delegados del lado del cliente y del servidor cumplen roles complementarios y forman los dos lados de la comunicación. Es importante que estos delegados no bloqueen el hilo por demasiado tiempo. En el caso de necesitar realizar una comunicación continua entre ambas partes, se debe abrir un puerto nuevo y realizar el resto de la comunicación en otro hilo.

El procedimiento de inicio de un protocolo es el siguiente:

1. El servidor envía un mensaje de tipo TransactionMessage que contiene un string con el nombre del protocolo.
2. El servidor llama al delegado del protocolo.
3. El cliente recibe el mensaje y busca el protocolo con el nombre indicado
4. El cliente llama al delegado del protocolo.
5. Los delegados de ambos lados realizan la comunicación libremente.

En ciertos casos, el envío de la solicitud de protocolo es suficiente para transmitir al cliente un mensaje propiamente dicho. En aquellos casos, el delegado del lado del servidor estará vacío y no se realizará una comunicación entre partes durante el protocolo.

6.3.3 Comunicación servidor-cliente:

Una vez realizado el descubrimiento del servidor, el cliente envía un mensaje de tipo IdentificationMessage con un nombre único que el servidor utilizará para identificarlo. Luego se crea un objeto RobotClient, que a su vez inicia un hilo dedicado a la transacción de mensajes y protocolos sobre el puerto 13000.

En este hilo se envía un mensaje de tipo KeepAliveMessage cada 1000ms (interrumpiéndose durante la ejecución de protocolos) que el servidor utiliza para saber si la conexión se ha caído. También se verifica si hay solicitudes de ejecución de protocolo pendientes en la cola y se ejecutan. Para no consumir recursos, una vez vaciada la cola, el hilo se bloquea hasta realizarse una nueva solicitud desde otro hilo del servidor.

Debido a la naturaleza asincrónica del sistema de transferencias, para realizar una solicitud se utiliza la clase TransactionData, que contiene un protocolo, un array de argumentos y el resultado devuelto por el delegado del protocolo. Para la solicitud se utiliza el método BeginTransaction del RobotClient, que toma como parámetro una instancia de TransactionData. Para obtener el resultado se puede comprobar el flagTransactionData.Finalized y extraer el resultado de TransactionData.Result. Para no consumir recursos en la comprobación se puede utilizar el método TransactionData.Wait(), que bloquea el hilo actual hasta haberse finalizado la transacción. Es importante recordar que TransactionData representa una transacción individual y no puede ser reusado; deberá ser reconstruido con los mismos parámetros.

7.0 Interfaz de usuario:

El usuario va a poder controlar al robot mediante el joystick y la cámara que posee el robot, y gracias a esto poder controlarlo libremente por el espacio en el que esté.

8.0 Modulo de medición de signos vitales:

8.0.1 Descripción general:

Este módulo de medición de signos vitales contiene un sensor MLX90614 y un oximetro Max30100 que nos proporcionan la temperatura del paciente, el pulso, el oxígeno en sangre y presión arterial.

8.1 Sensor de temperatura:

Imagen 26- MLX90614 Sensor de temperatura

8.1.1 Descripción general:

El Sensor de temperatura infrarrojo MLX90614 fabricado por la empresa Melexis permite medir la temperatura de un objeto a distancia (sin contacto). El Sensor MLX90614 es un chip de silicio con una fina membrana micromecanizada, diseñada para ser sensible a la radiación infrarroja emitida por un objeto a distancia. El sensor posee internamente una etapa de amplificación y digitalización (ADC) de la señal procedente de la membrana. La salida del sensor es lineal y se compensa de acuerdo a las variaciones de la temperatura ambiente.

8.1.2 Diagrama de conexión:

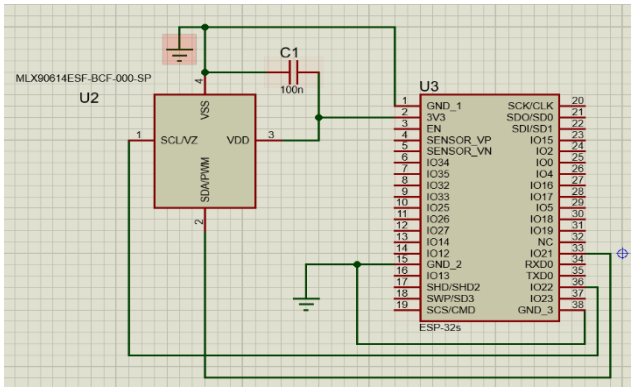


Imagen 27- Esquema del sensor de temperatura

8.1.3 Funcionamiento:

Según la ley de Stefan-Boltzmann, todo objeto por encima del cero absoluto (ºK) emite radiación cuyo espectro es proporcional a su temperatura. El MLX90614 recoge esta radiación y su salida es una señal eléctrica proporcional a la temperatura de todos los objetos en su campo de visión. Internamente el MLX90614 está constituido con un chip de silicio con una fina membrana micro mecanizada sensible a la radiación infrarroja, junto con la electrónica necesaria para amplificar y digitalizar la señal y calcular la temperatura. El MLX90614 viene calibrado de fábrica en un amplio rango de temperaturas: -40 a 85 °C para la temperatura ambiente y -70 a 382 °C para la temperatura de objetos. La precisión estándar es de 0.5 °C referente a la temperatura ambiente, aunque existen versiones médicas que ofrecen una resolución de 0.1ºC en temperaturas entre 35-38ºC.

8.1.4 Características:

- Módulo: GY-906
- Chip sensor: MLX90614ESF-BAA
- Voltaje de operación: 3.3V-5V DC
- Protocolo de comunicación SMBUS (subconjunto del I2C)
- Rango de temperatura ambiente de trabajo: -40℃ hasta +170℃
- Rango de temperatura de objeto: -70℃ hasta +380℃
- Precisión: ±0.5°C
- ADC interno de 17 bits
- Procesador digital de señal interno
- Regulador de voltaje 3.3V en placa
- Resistencias Pull-up a VIN en placa
- No necesita componentes adicionales
- Dimensiones: 16*11*6 mm
- Peso: 2.80 gramos

8.1.5 Programación:

```
import ustruct, time, network
from machine import I2C, Pin

class SensorBase:

    def read16(self, register):
        data = self.i2c.readfrom_mem(self.address, register, 2)
        return ustruct.unpack('<H', data)[0]

    def read_temp(self, register):
        temp = self.read16(register);
        # apply measurement resolution (0.02 degrees per LSB)
        temp *= .02;
        # Kelvin to Celcius
        temp -= 273.15;
        return temp;

    def read_ambient_temp(self):
        return self.read_temp(self._REGISTER_TA)

    def read_object_temp(self):
        return self.read_temp(self._REGISTER_TOBJ1)

    def read_object2_temp(self):
        if self.dual_zone:
            return self.read_temp(self._REGISTER_TOBJ2)
        else:
            raise RuntimeError("Device only has one thermopile")

    @property
    def ambient_temp(self):
        return self.read_ambient_temp()

    @property
    def object_temp(self):
        return self.read_object_temp()

    @property

    def object2_temp(self):
        return self.read_object2_temp()

class MLX90614(SensorBase):

    _REGISTER_TA = 0x06
    _REGISTER_TOBJ1 = 0x07
    _REGISTER_TOBJ2 = 0x08

    def __init__(self, i2c, address=0x5a):
```

```
        self.i2c = i2c
        self.address = address
        _config1 = i2c.readfrom_mem(address, 0x25, 2)
        _dz = ustruct.unpack('<H', _config1)[0] & (1<<6)
        self.dual_zone = True if _dz else False

i2c = I2C(scl=Pin(22), sda=Pin(21), freq=100000)
sensor = MLX90614(i2c)
temperaturas = []

station = network.WLAN(network.STA_IF)
station.active(True)
station.connect("Fibertel WiFi411 5.8GHz", "0141847689")
station.isconnected()
station.ifconfig()

print(sensor)
for i in range (15):
  temperaturas.append(round(sensor.read_object_temp(),2) +3)
  print(" Temperatura de la persona: ", round(sensor.read_object_temp(),2) +3  , "°C" )
  time.sleep_ms(1000)

contador = 0

for i in temperaturas:
  if contador == 0:
    contador = contador + 1
    pass
  else:
    temperaturas[0] = temperaturas[0] + i


temperature = temperaturas[0] / 15

if temperature > 38:
   print("Usted tiene fiebre y tu temperatura es ", temperature, "°C")
else:
   print("Usted está bien y tu temperatura es ", temperature, "°C")
```

Programación 5- Sensor de temperatura

8.1.6 Esquemático:

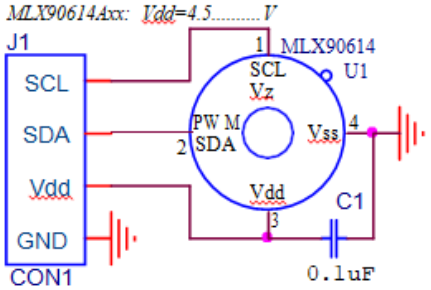


Imagen 28- Esquema del sensor de temperatura

8.2 Oximetro:

Imagen 29- Oximetro

8.2.1 Descripción general:

El MAX30100 es una solución integrada de sensor de pulsioximetría y monitor de frecuencia cardíaca. Combina dos LED, un fotodetector, óptica optimizada y procesamiento de señales analógicas de bajo ruido para detectar señales de pulsioximetría y frecuencia cardíaca. El MAX30100 funciona con fuentes de alimentación de 1.8V y 3.3V y se puede apagar mediante software con una corriente de espera insignificante, lo que permite que la fuente de alimentación permanezca conectada en todo momento.

8.2.2 Diagrama de conexión:

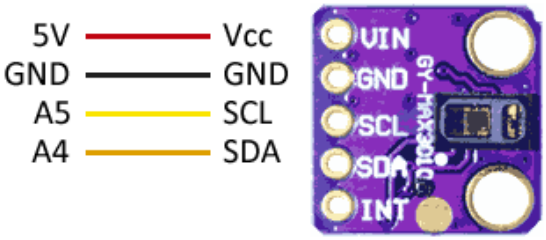


Imagen 30 – Conexiones del oximetro

8.2.3 Funcionamiento:

El MAX30100 tiene un LED rojo, un LED infrarrojo y un fotodetector en la parte superior del encapsulado. El LED rojo se emplea para oximetría de pulso, mientras que el LED infrarrojo se utiliza para medir el ritmo cardiaco. Como cada LED emite luz en el dedo de la persona, el fotodetector detecta las variaciones en la luz por cambios de volumen de la sangre.

Un convertidor de analógico a digital (ADC) de 16 bit con ratio de muestra programable cambia la salida del fotodetector en un valor digital. Este valor lo analiza posteriormente un procesador de señal analógico de bajo ruido on-chip que establece el ritmo cardiaco y el contenido de oxígeno en sangre. Los datos se leen mediante una interfaz I2C serie.

8.2.4 Caracteristicas:

- Consume muy poca energía (opera desde 1.8V y 3.3V).
- Corriente de apagado ultrabaja (0.7 µA, típico).
- Capacidad de salida de datos rápida.
- Voltaje de Operación: 5V DC
- Regulador de voltaje de 3.3V y 1.8V en placa
- Led rojo de 660nm
- Led infrarrojo de 920nm
- Filtro de luz entre 50 y 60Hz
- Protocolo de comunicación: I2C
- ADC delta sigma de hasta 16 bits
- Temperatura de trabajo: -40°C hasta +85°C
- Dimensiones: 14mm x 17mm
- VIN: 5V DC
- SCL: I2C CLOCK
- SDA: I2C DATA
- INT: Interrupción, activo a estado bajo
- IRD: cátodo led infrarrojo (No conectar)
- RD: cátodo led rojo (No conectar)
- GND: 0V

8.2.5 Programación:

```
from lib.MAX30102 import MAX30102
from machine import sleep
from utime import ticks_diff, ticks_ms
import logging


logging.basicConfig(level=logging.INFO)

# Sensor instance. If left empty, loads default ESP32 I2C configuration
sensor = MAX30102()


# The default sensor configuration is:
# Led mode: 2 (RED + IR)
# ADC range: 16384
# Sample rate: 400 Hz
# Led power: maximum (50.0mA - Presence detection of ~12 inch)
# Averaged samples: 8
# pulse width: 411
print("Setting up sensor with default configuration.", '\n')
sensor.setup_sensor()

sleep(1)

# The readTemperature() method allows to extract the die temperature in °C
# print("Reading temperature in °C.", '\n')
# print(sensor.readTemperature())

# Select wether to compute the acquisition frequency or not
compute_frequency = False

print("Starting data acquisition from RED & IR registers...", '\n')
sleep(1)

t_start = ticks_ms()    # Starting time of the acquisition
samples_n = 0           # Number of samples that has been collected

while(True):
    # The check() method has to be continuously polled, to check if
    # there are new readings into the sensor's FIFO queue. When new
    # readings are available, this function will put them into the storage.
    sensor.check()

    # Check if the storage contains available samples
    if(sensor.available()):
        # Access the storage FIFO and gather the readings (integers)
        red_reading = sensor.popRedFromStorage()
        IR_reading = sensor.popIRFromStorage()
```

```
# Print the acquired data (can be plot with Arduino Serial Plotter)
print(red_reading, ",", IR_reading)

# We can compute the real frequency at which we receive data
if (compute_frequency):
    samples_n=samples_n+1
    if ( ticks_diff(ticks_ms(), t_start) > 999 ):
        f_HZ = samples_n/1
        samples_n = 0
        t_start = ticks_ms()
        print("acquisition frequency = ",f_HZ)
```

Programación 6- Oximetro

8.2.6 Esquemático:

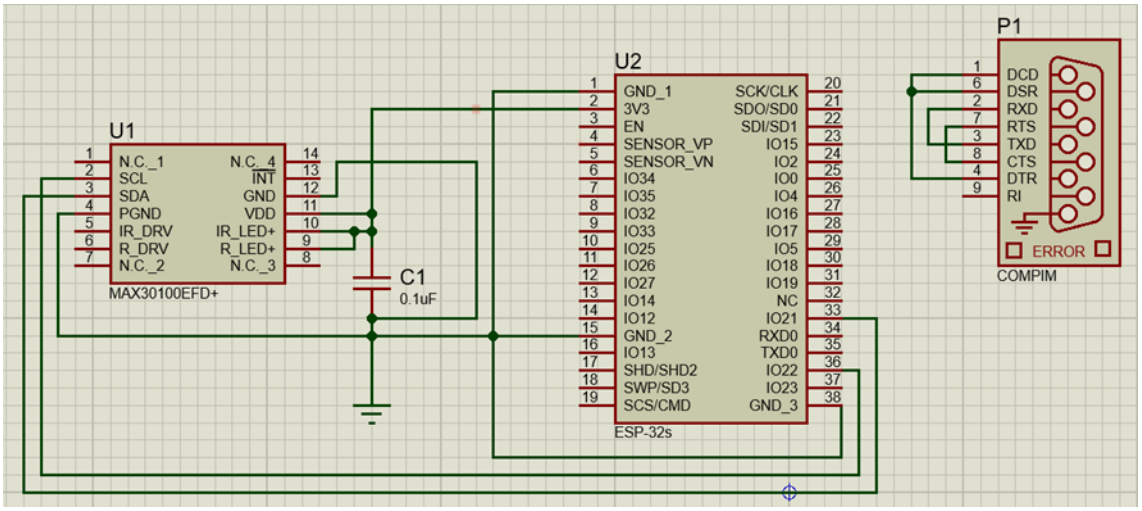


Imagen 31- Esquema del oximetro

8.3 Display OLED:

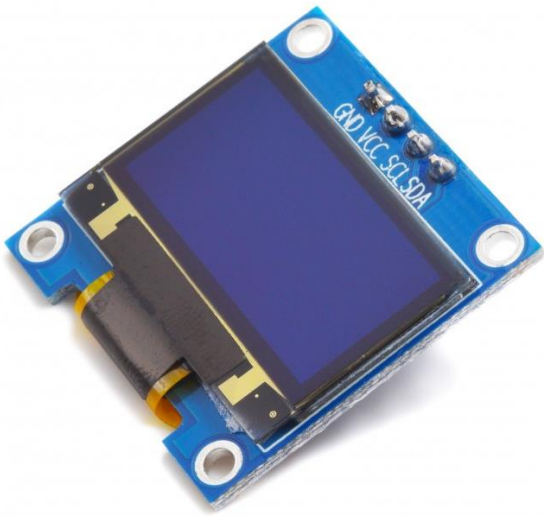


Imagen 32- Display OLED

8.3.1 Descripción:

OLED (diodos emisores de luz orgánicos) es una tecnología de emisión de luz plana, que se fabrica colocando una serie de películas delgadas orgánicas entre dos conductores. Cuando se aplica corriente eléctrica, se emite una luz brillante. Los OLED son pantallas emisoras que no requieren luz de fondo y, por lo tanto, son más delgadas y más eficientes que las pantallas LCD (que requieren luz de fondo blanca). A diferencias de las pantallas LED o LCD, que usan la retroiluminación general del panel de píxeles, estas pantallas cuentan con un diodo emisor de luz por cada píxel.

9.0 Modulo de medición de dióxido de carbono:

9.0.1 Descripción general:

Este módulo integrado al robot nos daría información en un display de cuanta ppm hay en el ambiente, y tiene también leds (rojo, amarillo y verde) y un buzzer que nos dirían si hay que ventilar el ambiente. Si el ambiente tiene buena ventilación se prendería el led verde y las ppm estarían en el rango de 450-580 ppm, si el ambiente no tiene muy buena ventilación se prendería el led amarillo y las ppm estaría entre 581- 640, si el ambiente esta con muy mala ventilación se prendería el led rojo y el buzzer (que actúa como una alarma auditiva), habría que ventilar el ambiente lo antes posible, las ppm superaran los 641 ppm

9.1 Sensor de calidad del aire:

Imagen 33- Conexión del MQ135

9.1.1 Descripción general:

El MQ-135 es un sensor de calidad del aire que permite detectar algunos gases peligrosos como Amoniaco, Dioxido de Nitrógeno, Alcohol, Benzeno, Dioxido y Monoxido de carbono. El sensor puede detectar concentraciones de gas entre 10 y 1000 ppm y es de utilidad para detección de gases nocivos para la salud en la industria principalmente.

9.1.2 Diagrama de conexión:

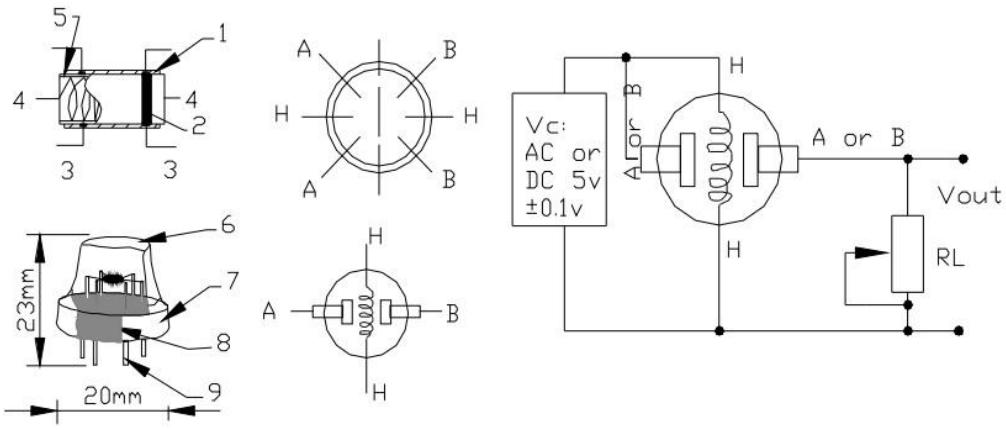


Imagen 34- Conexión del MQ135

9.1.3 Características:

- Voltaje de operación: 5V DC
- Corriente de operación: 150mA
- Potencia de consumo: 800mW
- Tiempo de precalentemiento: 20 segundos
- Resistencia de carga: Potenciómetro (Ajustable)
- Detección de partes por millón: 10ppm~1000ppm
- Concentración detectable: Amoniaco, sulfuro, benceno, humo
- Concentración de oxígeno: 2%~21%
- Humedad de operación: <95%RH
- Temperatura de operación: -20°C~70°C

9.1.4 Funcionamiento:

El sensor posee un microtubo de cerámica con óxido de aluminio (Al2O3), una capa sensible de dióxido de estaño (SnO2), el electrodo de medida y el calentador, que están fijados en su estructura. Este último proporciona las condiciones necesarias para activar el sensor, el mismo no nos mostrará directamente las ppm del gas, sino que devolverá una cantidad proporcional a la resistencia que producirá cierta cantidad del gas. Estos datos son tomados mediante cuatro de los seis pines que el dispositivo posee, los dos restantes son para obtener la energía necesaria para calentar el microtubo.

9.1.5 Programación:

```
#include <LiquidCrystal.h>
int sensorValue;
const int rs = 12, en = 11 , d7 = 7 , d6 = 8 , d5 = 9, d4 = 10;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
int redLed = 6;
int yellowLed = 5;
int greenLed = 4;
int buzzer = 3;


void setup(){
  pinMode(redLed, OUTPUT);
  pinMode(greenLed, OUTPUT);
  pinMode(yellowLed, OUTPUT);
  pinMode(buzzer, OUTPUT);

  lcd.begin(16, 2);
  Serial.begin(9600);                          // sets the serial port to 9600
 }

void loop(){
  sensorValue = analogRead(0)+ 320 ;       // read analog input pin 0
  Serial.print("AirQua=");
  Serial.print(sensorValue, DEC);               // prints the value read
  Serial.println(" PPM");
  lcd.setCursor(0,0);
  lcd.print("ArQ=");
  lcd.print(sensorValue ,DEC);
  lcd.print(" PPM");
  lcd.println("       ");
  lcd.print("  ");

  if (sensorValue > 600)  // hay peligro de covid
  {
    digitalWrite(redLed, HIGH);
    lcd.setCursor(0, 1);
    lcd.print("Peligro!!!");
    digitalWrite(greenLed, LOW);
    digitalWrite(yellowLed, LOW);
    tone(buzzer, 1000, 200);
  }
  else if ( sensorValue > 550 and sensorValue < 600){
    digitalWrite(yellowLed, HIGH);
    lcd.setCursor(0, 1);
    lcd.print("Normal!");
    digitalWrite(redLed, LOW);
    digitalWrite(greenLed, LOW);
  }
  else {
```

```
    digitalWrite(greenLed, HIGH);
    lcd.setCursor(0, 1);
    lcd.print("Bien!");
    digitalWrite(redLed, LOW);
    digitalWrite(yellowLed, LOW);
  }

  delay(1000);                                    // wait 100ms for next reading
  lcd.clear();

}
```

Programación 7- Sensor Mq135

9.1.6 Esquemático:

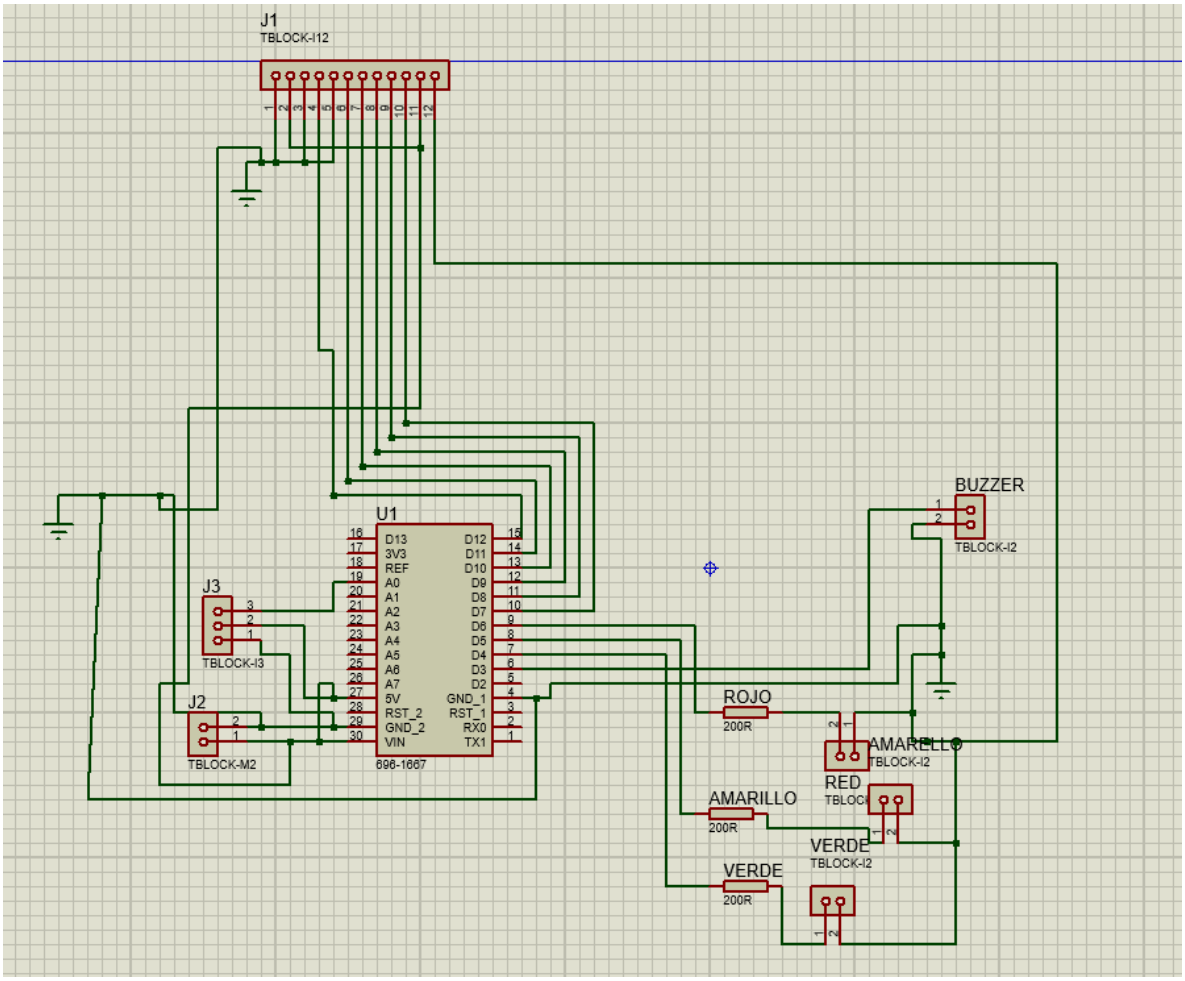


Imagen 35- Esquemático del MQ135

10.0 Información de trabajo:

10.1 Condiciones de entorno:

Trabajado en los sectores de:

- Taller de avionica 2.
- Taller de procedimientos.

10.2 Instrumental utilizado:

- Multimetro
- Osiloscopio portatil
- Torno taladro de mano y de banco.
- Pistola de siclicona
- Remachadora
- Destornilladores
- Pinzas
- Morza

10.3 Bibliografía

- MQ135 Datasheet: https://www.olimex.com/Products/Components/Sensors/Gas/SNS-MQ135/resources/SNS-MQ135.pdf
- MLX90614 Datasheet: Downloads/MLX90614-Datasheet-Melexis.pdf
- MAX30100 Datasheet: https://datasheets.maximintegrated.com/en/ds/MAX30102.pdf
- ESP32Datasheet: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
- HC-SR04 información: https://lastminuteengineers.com/arduino-sr04-ultrasonic-sensor-tutorial/
- HC-SR04 Datasheet: http://www.datasheet.es/PDF/779948/HC-SR04-pdf.html
- L298N Datacheet: https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf
- Database: https://randomnerdtutorials.com/esp32-esp8266-mysql-database-php/
- ESP32 Wifi: https://www.digikey.com/es/articles/how-to-select-and-use-the-right-esp32-wi-fi-bluetooth-module