ESCUELA DE EDUCACION SECUNDARIA TECNICA N° 5 “2 DE ABRIL” – TEMPERLEY – BUENOS AIRES

# Robot explorador Tabibito

MATERIA : PROYECTO Y DISEÑO ELECTRONICO

FECHA : 4/4/2024-15/11/2024


AUTORES:

Originales:
Lucero Stefano Stefxyz01@gmail.com
Yolli Benjamin benjayolli155@gmail.com
Colaboradores/agregados:
Astrada Bruno astradatobias33@gmail.com
Cionci Manuel cioncimanuelantonio@gmail.com
Aguero Thiago thiagotomasaguero22@gmail.com
Blasques Mayron silvamayron13@gmail.com
Jofre Lautaro lautarojofre999@gmail.com
Apolinario Fabrizio apolinariofabrizio216@gmail.com
Aramayo Aaron ociel.aaron.aramayo@gmail.com


PROFESOR: ING. MARTIN LEGUIZAMON[i]

1 Repositorio de Github

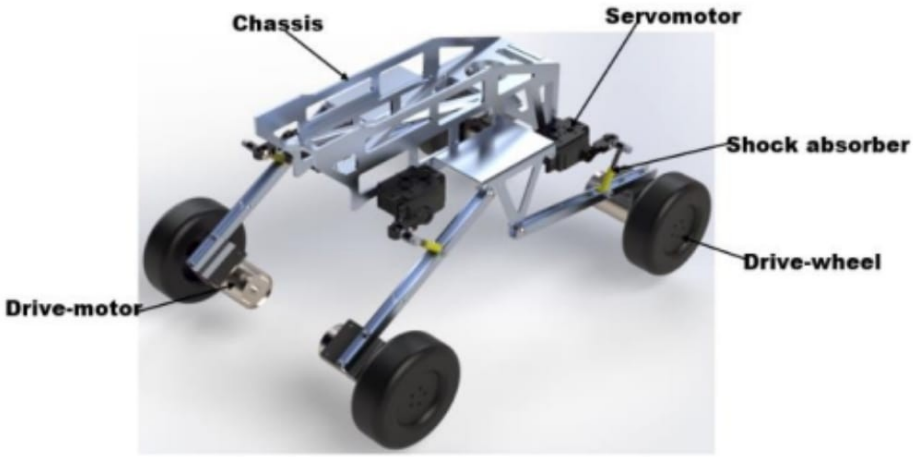
# Indice del proyecto Robot explorador Tabibito


Descripción del proyecto.... 4
Diagramas en bloques.... 4
Armado.... 5
Introducción Teórica.... 7
Explicación de la estabilización.... 8
Explicación seccionada de la placa.... 14
Alimentación y reguladores:.... 14
fuente step-down.... 14
Principio de funcionamiento.... 15
Ventajas de las fuentes step-down.... 15
Aplicaciones comunes.... 16
AMS1117.... 16
Características principales del AMS1117.... 16
Pinout del AMS1117.... 16
Cómo funciona internamente.... 17
Aplicaciones típicas.... 17
Diseño del circuito con AMS1117.... 17
Ejemplo de conexión para AMS1117-3.3 (3.3 V fijo).... 17
AMS1117 Ajustable (AMS1117-ADJ).... 18
Ventajas y limitaciones.... 18
Ventajas:.... 18
Limitaciones:.... 18
Programador:.... 18
Componentes clave:.... 19
Cómo usarlo:.... 19
Modulo/Microcontrolador.... 19
Características principales:.... 19
Cómo funciona:.... 20
Modulo reloj:.... 20
Módulo RTC DS1302: Cómo funciona.... 20
Características principales.... 20
Cómo funciona.... 21
Uso típico.... 21
Ventajas.... 21
Los sensores.... 21
1. Sensor de gas.... 22
Informe: Funcionamiento de los Sensores de Gas MQ.... 22
Introducción.... 22
Estructura y Principio de Funcionamiento.... 22
Modelos y Gases Detectados.... 22
Calibración.... 23
Ventajas y Limitaciones.... 23
Aplicaciones Prácticas.... 23
Conclusión.... 23
2. Sensor de distancia ultrasonido.... 23
Proceso de Medición:.... 24
Cálculo de Distancia:.... 24
Limitaciones:.... 24
3. Sensor de presión atmosférica.... 24
Funcionamiento del BMP180.... 24
Características técnicas principales.... 25
4. Sensor de temperatura y humedad.... 25
Características Comunes.... 25

DHT11 25
Aplicaciones Comunes 25
Rs485 26
1. Introducción al RS-485 26
2. Características Técnicas 26
3. Ventajas del RS-485 26
4. Aplicaciones del RS-485 27
5. Limitaciones y Consideraciones 27
Placa de Movimiento con mega 27
El módulo BTS7960 28
1. Características principales 28
2. Estructura del módulo 28
3. Funcionamiento básico 28
4. Conexión típica 29
5. Ventajas del BTS7960 29
6. Ejemplo de aplicación 29
Giroscopio MPU6050 29
1. Características principales 29
2. Ventajas 30
3. Componentes en el módulo físico 30
4. Aplicaciones comunes 30
5. Funcionamiento básico 30
Modulo controlador de servos PCA9685: 30
¿Qué es el módulo PCA9685? 30
Características principales 31
¿Cómo funciona? 31
1. Comunicación I2C 31
2. Generación de PWM 31
3. Frecuencia 31
4. Control de servomotores 31
Conexiones típicas 31
Pcb de las placas 32
Placa motores 32
Placa sensores 33
Software 33
Diagrama de gantt 45
Presupuesto total y parcial del proyecto 45

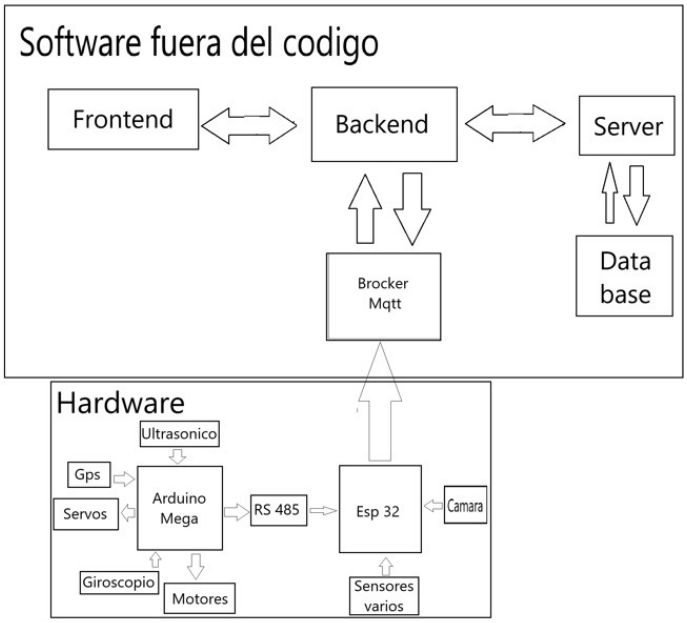
3 Repositorio de Github

## ➢ Descripción del proyecto




para explicar de forma completa el proyecto tenemos que entender cual es el propósito del robot, este robot fue ideado con el fin de poder desplazarse en terrenos irregulares, poder tomar registro de su entorno como grabaciones del encontrado, este registro lo debería hacer una cámara instalada en la parte superior del robot que debería grabar y subir a su base de datos donde se debe hacer el seguimiento de la información recolectada. A pesar de que el robot sea autónomo nosotros lo habíamos planteado que una persona este revisando la computadora donde esta conectado el robot para poder ver la información recolectada.
El proyecto también cuenta con un sistema de auto nivelación que es lo que loe permite adaptarse a terrenos irregulares variando la altura de una para u otra dependiendo la inclinación del robot, esta función es importante para que el robot pueda desplazarse en pendientes o en terrenos rocosos sin dificultad.

### *Diagramas en bloques*

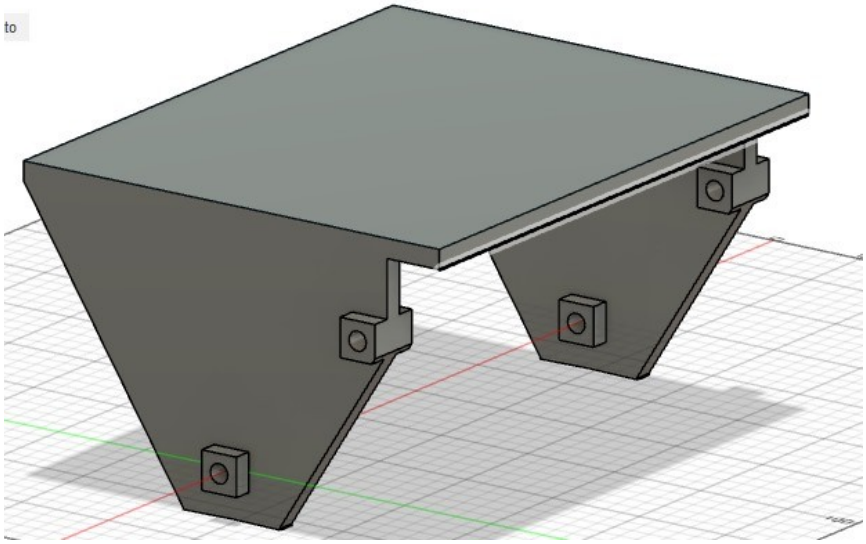



4 Repositorio de Github

## ➢ Armado

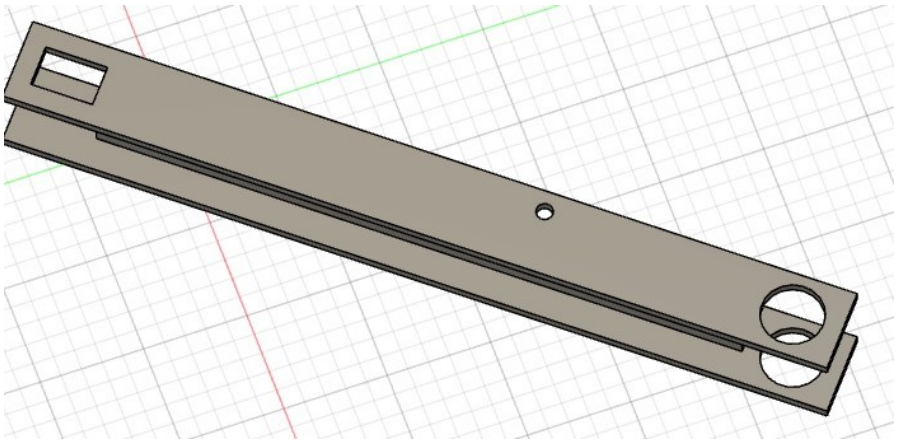
Para explicar el proyecto en un mayor detalle lo idea es que sepan como este esta armado,si bien en la imagen anterior esta explicado esta demasiado desligado por lo que vamos a ir paso por paso

A esta primer parte la vamos a llamar base y tiene la siguiente forma



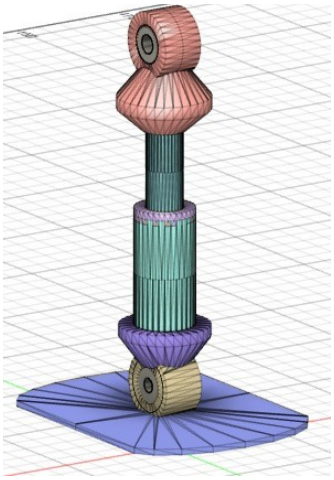
Esta base es la que une la carcasa principal y las patas

La siguiente parte son las patas las cuales como veremos a continuación tiene una forma simple



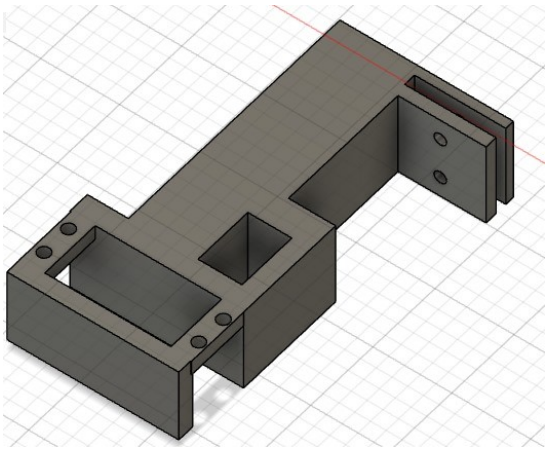
Lo único a destacar es los 3 agujeros que tiene,el circular grande el que une la base junto a estas patas,el segundo la mas pequeño es donde va el tornillo para el amortiguador el cual veremos a continuación y el ultimo agujero es donde van nuestros motores

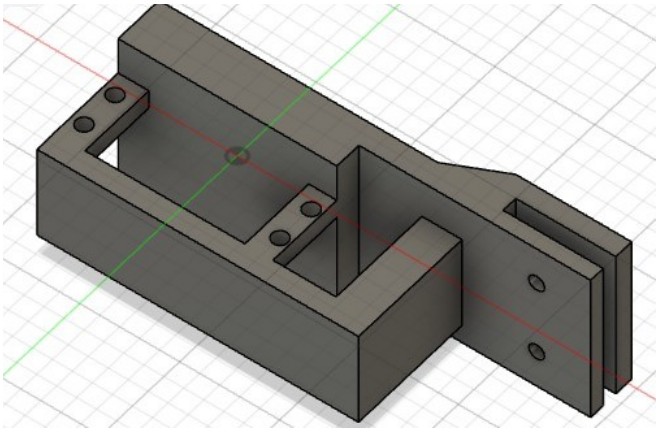
Ahora pasemos con los amortiguadores



No tiene nada destacable aparte de que para poder poner esto necesitamos los servomotores y para poner los servomotores necesitamos unos soportes los cuales veremos ahora

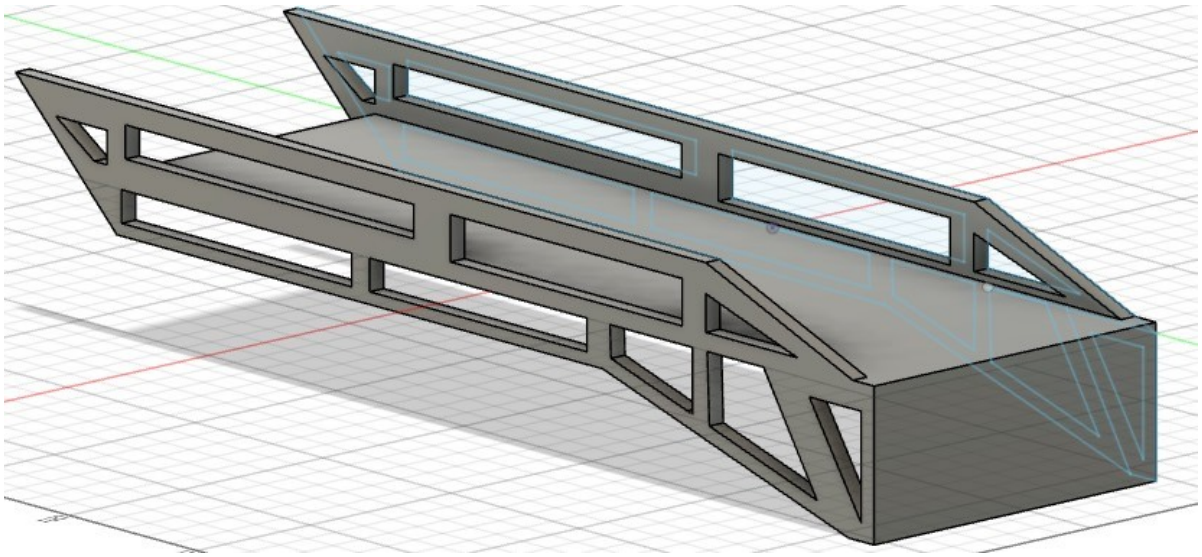
Los Soportes de servos son así





Estos como su nombre indica actúan como soporte para los servomotores y tiene seis agujeros circulares los 2 pares que están junto a los rectángulos son los agujeros para los tornillos del servomotor y el otro par de agujeros es para poner los soportes a la base que mostramos anteriormente
Como ven tenemos dos modelos distintos de soportes la razón de esto es que las patas como pueden ver en nuestra imagen inicial tiene las dos patas en distintos lugares y para que estos se puedan estabilizar tienen que estar a distancias distintas,la razón por la que son de esta forma la explicaremos mas adelante

y por ultimo quedaría nuestro cuerpo



en este irán casi todos los componentes que harán mover el robot,por lo que para ver bien como funcionara esto vamos a ver primero una:

---

## ➢ Introducción Teórica

Para introducir la teoría que tiene el robot primero tenemos que ver bien como es el funcionamiento si bien antes lo dijimos no fue muy preciso por lo que vamos a detallar un poco mas. Este robot para moverse utiliza programación en arduino utilizando el microcontrolador Arduino mega, la razón de su movimiento es que a través de un componente que actuá como GPS le damos ordenes de que el robot vaya a 5 localizaciones las cuales pondremos en forma de coordenadas con esto el robot sabrá a donde tendrá que moverse, ademas de esto también tendrá 3 sensores ultrasonicos para que evite chocarse con cualquier obstáculo que pueda molestar al robot,mientras el robot se va moviendo va a ir recopilando datos del entorno como podrían ser temperatura y humedad,presión,etc,ademas durante el transcurso del movimiento este posiblemente haya bastantes desniveles en el transcurso hacia las 5 localización asi que para evitar que estos desniveles puedan afectar negativamente al robot realizamos un sistema de estabilización.

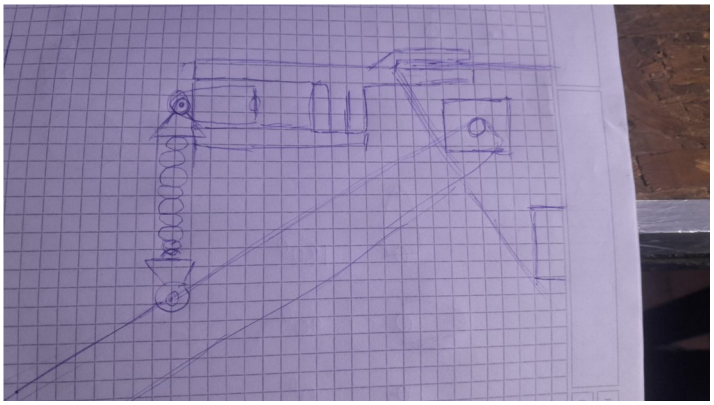
Esta estabilización es realizada a través de los servomotores y los amortiguadores,esa es la razón por la que como mencionamos antes los soportes tienen diferente forma,ademas de tener esta diferente forma las patas traseras tienes in agujero para el amortiguador de 4cm mas lejos que el de las delanteras,vamos a detallar un poco esto

Primero hay que entender como se realiza esta estabilización y lo podemos ver en la siguiente imagen

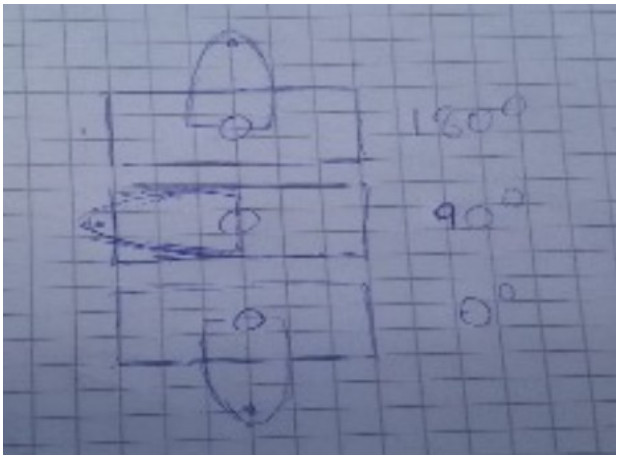
---

# ➢ Explicación de la estabilización

Como pueden ver el servomotor es el que realiza todo por lo que este es el mas importante la pala que tiene este tiene una variación máxima de 2 cm ya que si intentamos que varié mas que eso el servomotor perderá mucha de su capacidad para aguantar el peso y puede que no aguante para la estabilización lo podemos ver bien en la siguiente imagen

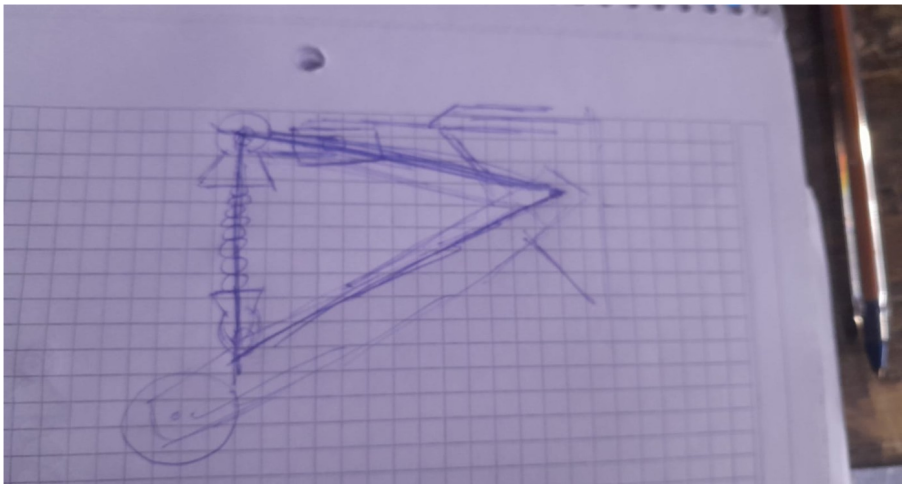


Como se ve si bien el servo varia 180° para calcularlo vamos a usar tres valores 180° 90° y 0° la razón es que con solo esos tres podemos sacar valores promedio para ubicar esto(cambiar)




Entendiendo esto ahora podemos ver las piezas tres piezas que son El soporte del servo, El amortiguador y las patas(lo que me refiero al mencionar las patas es el agujero que tenemos que realizar para que la estabilización sea precisa ya que el amortiguador tiene que estar unida a esta pata pero a una distancia que mostraremos a continuación)ahora vamos a ver como realizar la estabilización

Para esto tenemos que pensar en cada uno de los componentes como si fuera el cateto de un triángulos como lo podemos ver en la siguiente imagen



ademas de esto hay que tener en cuenta que el calculo lo tendremos que hacer dos veces ya que como mencionamos anteriormente la distancia entre las patas traseras y delanteras son distintas

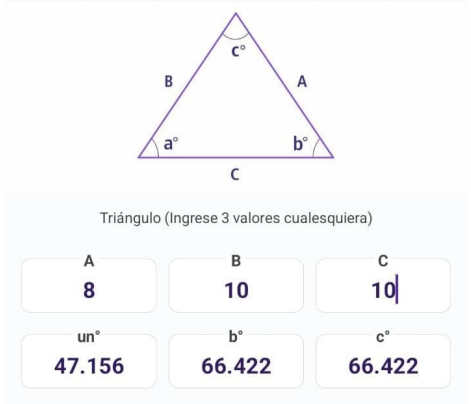
Por lo que empezaremos con las patas delanteras
Si bien ya están las piezas con sus respectivas medidas vamos a empezar dejando eso de lado
No tenemos ningún valor lo único que sabemos es que la variación máxima de ángulos es de unos 60 – 90° ya que mas que ello es irreal que el robot intente pasar por ahí pues o se juntarían las patas o haría que la base del robot choque con el piso
La primer pieza que yo realice en base a esto fue los amortiguadores en el primer diseño ya que no había tenido en cuenta lo antes mencionado tenia una medid de unos 15cm tras tenerlos en cuenta el amortiguador que realice es de unos 8cm también se podría buscar que sea aun mas chico para una mayor variación de ángulos sin embargo podría hacer que su resistencia baje demasiado por lo que me quede con.

Una vez conseguidos estos valores lo mejor que podía hacer era realizar prueba y error hasta encontrar valores que sean lo suficientemente amplios para que nos den los ángulos antes mencionados,al hacer esto los valores que me dio para el soporte y las patas fueron de 10cm ambos(cabe recalcar que si bien se menciona el soporte lo que se utiliza es la conexión entre el amortiguador y el servo siendo la punta de la pata del servo,esto es importante ya que es la variación de 2cm que mencionamos anteriormente de no ser así no habría variación por lo tanto no habría estabilización,también recalcar que al moverse el servo motor hará que la distancia del amortiguador también varié indirectamente)

Los cálculos que realice fueron a través de la aplicación AngleCal y los podemos ver a continuación

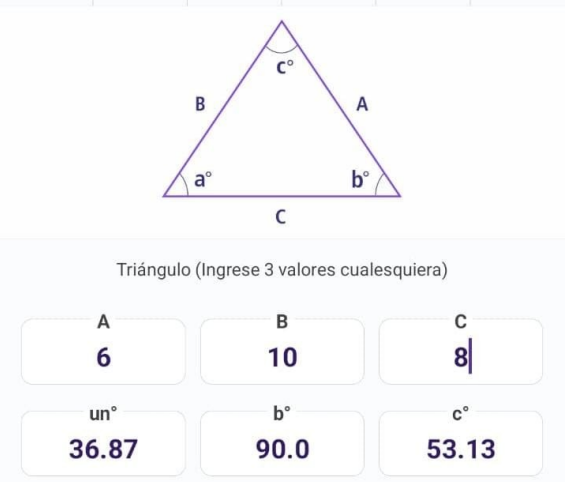
Como podemos ver en las imágenes a continuación tenemos 6 valores 3 en forma de ° y los otros 3 en M lo que nos interesa son los 3 valores en M y valor en ° llamado un°,vamos a empezar la explicación

imagen 1



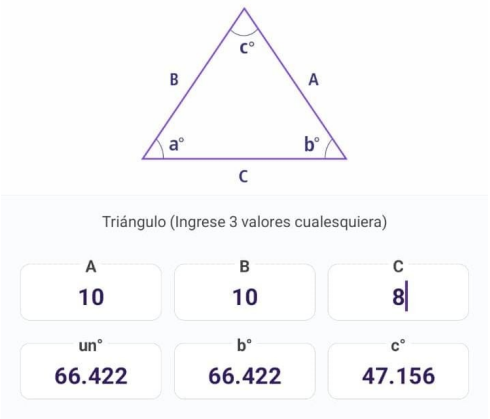
La medida A es el valor del amortiguador el cual mide como mencionamos antes 8cm
La medida B es el agujero de la pata que mide 10cm
y La medida C es 10 ya que el servo motor que en este caso esta a 90°
Dándonos que el valor de un° también conocido como a° es de 47°
osea que cuando el valor de el servomotor este a 90° jara que el angulo de la pata este a 47°

Imagen 2



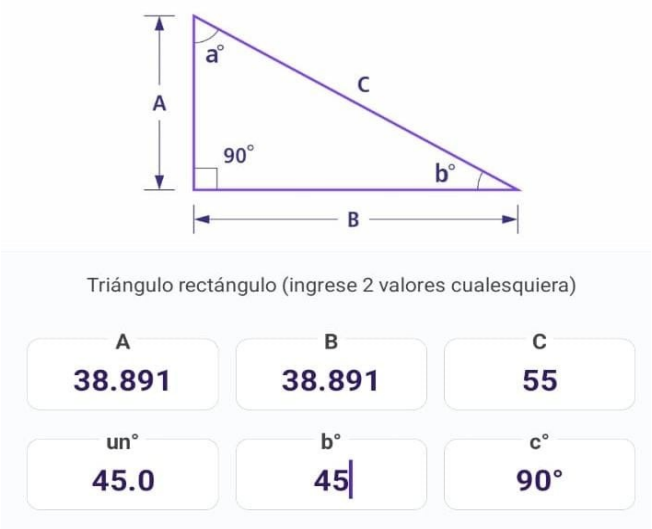

La medida A es el valor del amortiguador es de 6cm la razón de que el amortiguador tenga esta medida no es que cambiamos el amortiguador ni que se achico si no que al estar el servo motor a 180° varia 2cm esos 2cm que varia hace que el amortiguador para los cálculos mida 6 aunque siga midiendo lo mismo
La medida B es el agujero de la pata que mide 10cm
y La medida C es 8 ya que el servo motor que en este caso esta a 180°
Dándonos que el valor de un° también conocido como a° es de 37°
osea que cuando el valor de el servomotor este a 180° jara que el angulo de la pata este a 37°

imagen 3




La medida A es el valor del amortiguador el cual mide 10cm la lógica es la misma que la mencionada en la imagen anterior sin embargo al se 0° en vez de 180° este valor se suma en vez de restar

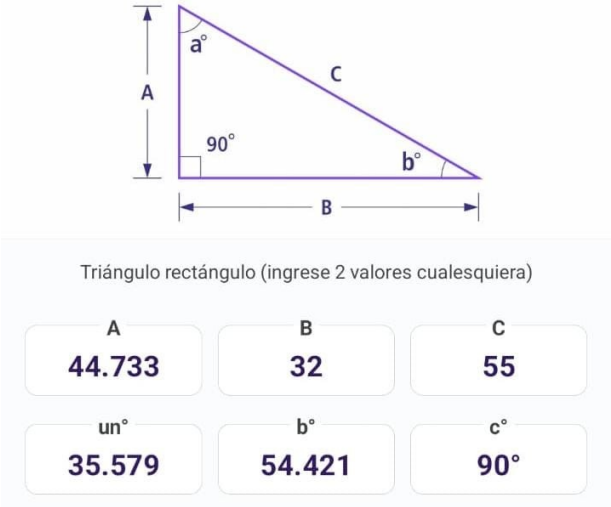
La medida B es el agujero de la pata que mide 10cm
y La medida C es 8 ya que el servo motor que en este caso esta a 0°
Dándonos que el valor de un° también conocido como a° es de 66°
osea que cuando el valor de el servomotor este a 0° jara que el angulo de la pata este a 66°
Osea que entre 0° y 180° tenemos una variación de casi 30°




Con esto hecho lo que quedaría seria hacer los cálculos para las otras patas sin embargo estas son mas fáciles,lo que tenemos en cuenta es la distancia entre el eje de la pata trasera y delantera siendo esta distancia de 7cm

Normalmente nuestra pata trasera va a estar a 45° esta pata tendrá una distancia de 55cm (son 40 de la pata + 15 del tamaño de las ruedas) con esto y teniendo en cuenta que sera un triangulo rectángulo podemos calcular a que distancia del suelo va a estar del suelo siendo de 39cm

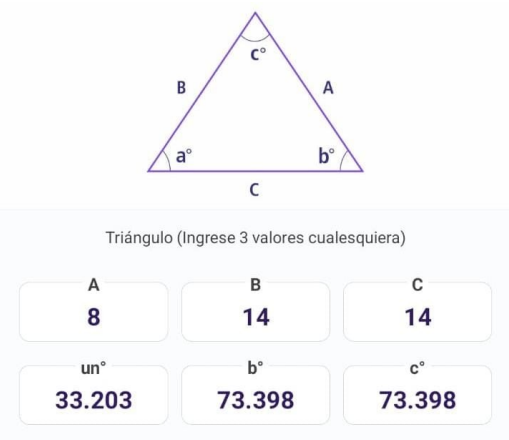
A estos 39cm les tenemos que restar los 7cm de la distancia entre ellos dándonos 32cm con eso ya podemos calcular el angulo que necesitaremos para cuando el angulo de las patas traseras estén a 45° siendo el calculo tal que así



Dándonos que si en la pata trasera esta a 45° la pata delantera tiene que estar a 35,6°(si bien el calculo dice 54,4° esto al ser el lado contrario de la pata no tenemos que tener en cuenta este valor si no el contrario a este siendo así 35,8°)

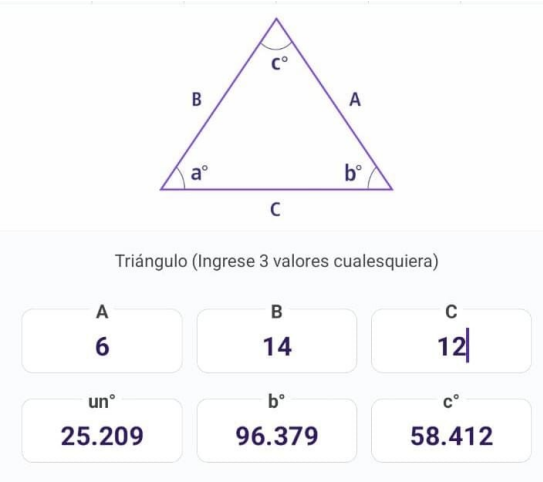
Ahora que tenemos esto necesitamos realizar las piezas de forma diferente,exceptuando el amortiguador,los cambios que realice fue aumentar la distancia del agujero de la pata en 4cm(total 14cm) y para el soporte lo mismo hacerlo de 14cm al estar a 90° el servo,estos valores fueron seleccionados ya que la variación de ángulos en este caso no tiene que ser la misma que en la parte trasera,esto si bien no hace que este a los 35,8° hace que este a algo cercano que es 33° como veremos a continuación

Otra vez se utiliza una lógica parecida a la anterior,vamos con lo cálculos
imagen 1




La medida A es el valor del amortiguador el cual mide 8cm
La medida B es el agujero de la pata que mide 14cm
y La medida C es 14 ya que el servo motor que en este caso esta a 90°
Dándonos que el valor de un° también conocido como a° es de 33°. osea que cuando el valor de el servomotor este a 90° jara que el angulo de la pata este a 33

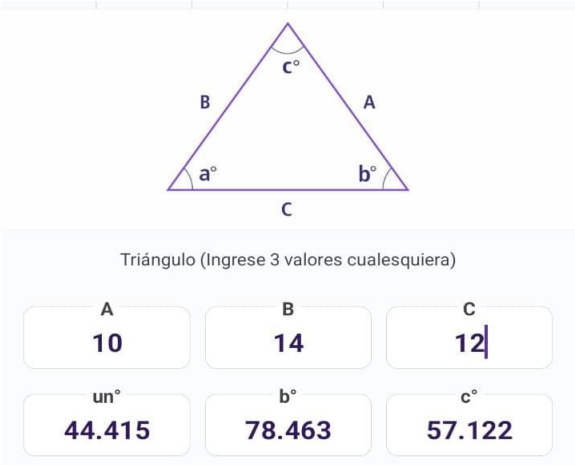
Imagen 2




La medida A es el valor del amortiguador es de 6cm la razón es la explicada en la anterior parte de los cálculos
La medida B es el agujero de la pata que mide 14cm

y La medida C es 12 ya que el servo motor que en este caso esta a 180°
Dándonos que el valor de un° también conocido como a° es de 25°
osea que cuando el valor de el servomotor este a 180° jara que el angulo de la pata este a 25°

imagen 3




La medida A es el valor del amortiguador el cual mide 10cm la lógica es la misma que la mencionada en la imagen anterior
La medida B es el agujero de la pata que mide 14cm
y La medida C es 12 ya que el servo motor que en este caso esta a 0°
Dándonos que el valor de un° también conocido como a° es de 44°
osea que cuando el valor de el servomotor este a 0° jara que el angulo de la pata este a 44°

Dándonos que la variación de los ángulos es de 20° a diferencia de los 30° en la pata trasera,puede que esta diferencia como se monstruo antes no sea completamente exacta ya que para eso debería de tener la situación de que al estar a 90° ambos servos el angulo de las patas deberían de ser 45° y 35,8° respectivamente sin embargo no es así por lo que de base tendrá un problema ahí,ese problema de base sera solucionado por el código que ya veremos mas adelante

con eso estaría toda la teoría detrás de la estabilización y la parte mecánica del robot por lo que quedaría la teoría detrás de el software que seria el siguiente

El robot cada 100ms tomara medidas a través de los sensores que mencionamos anteriormente,la razón por la que hará estas mediciones cada esta cantidad de tiempo es que si lo hiciera todo el rato retrasaría las acciones que debe hacer los servomotores para la estabilización,no hay ninguna por la que se utilice este tiempo y no uno mayor o menor,simplemente no afecta demasiado a los servomotores y los datos que obtenga en 10o ms no deberían cambiar mucho,como decíamos estos datos los ira recopilando y los enviara a una base de datos en la cual se almacenaran todos estos datos para lo que se necesiten,los datos son visibles a través de una pagina web la cual puedes utilizar para ver en tiempo real la localización actual del robot y la cámara que tiene de forma frontal,eso

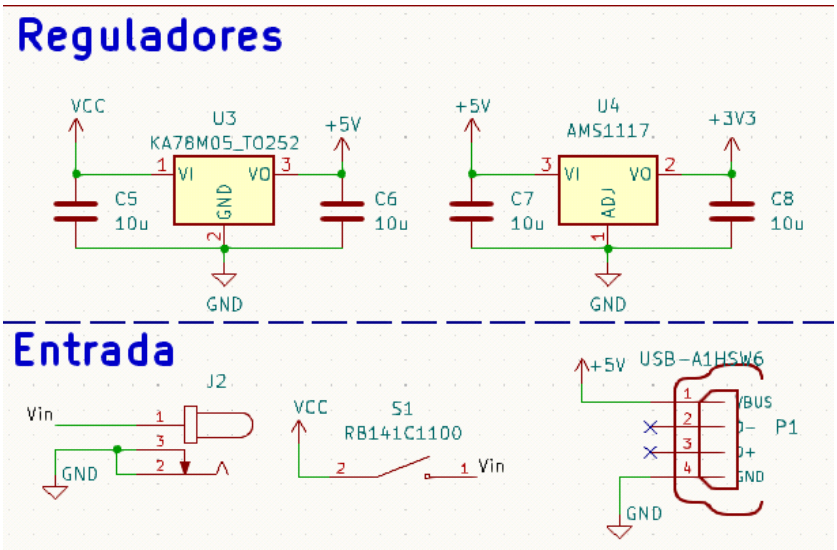
seria de forma simple el como es la teoría del software para verlo en mayor detalle puede ir a la pagina …. la cual tiene toda la explicación del código de este robot

---

## ➢ Explicación seccionada de la placa

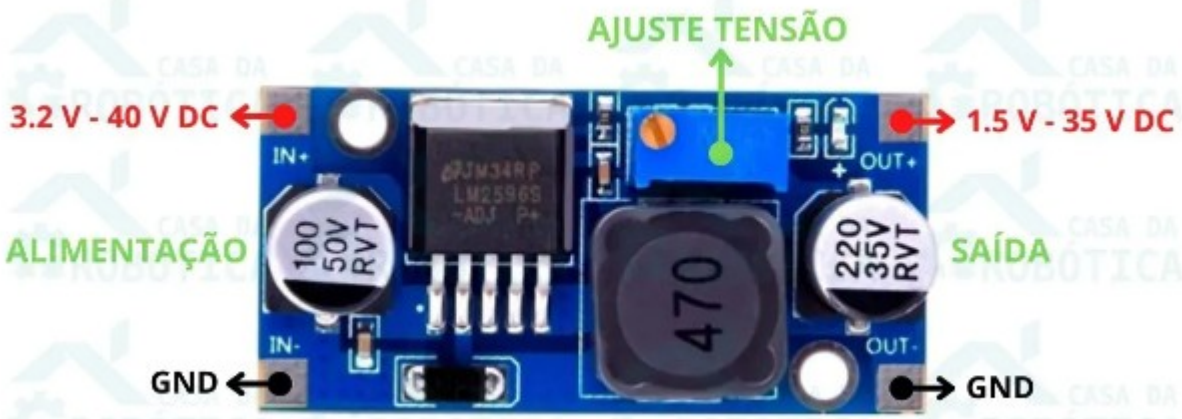
### ➢ *Alimentación y reguladores:*

En este apartado tenemos una entrada de alimentación que va directamente a la batería de 22,2v y 28,80Ah. Es una batería de lipo, para evitar daños en las placas encontramos una tecla de on off que nos permite darle un corte de emergencia al circuito en caso de que el circuito falle, esto lo hacemos para evitar que la placa se siga dañando. Por otro lado tenemos los 2 reguladores de tensión que bajan esos 22v en 5 y en 3v3, para describir de forma general los reguladores de tensión buscamos lo que nos puede decir el propio creador en el datasheet.




## fuente step-down

Una **fuente step-down**, también conocida como **convertidor buck**, es un circuito electrónico que convierte un voltaje de entrada mayor en un voltaje de salida menor,




manteniendo una alta eficiencia energética. Este tipo de fuente se utiliza cuando es necesario reducir el voltaje para alimentar dispositivos que funcionan a niveles de voltaje más bajos.

## Principio de funcionamiento

El convertidor buck utiliza un interruptor (generalmente un transistor), un diodo, un inductor y un capacitor para realizar la conversión. Su funcionamiento básico se puede describir en dos fases:

1. **Fase de encendido (Switch cerrado):**
    - El interruptor se cierra, lo que permite que la corriente fluya desde la fuente de alimentación a través del inductor hacia la carga.
    - Durante este tiempo, el inductor almacena energía en forma de campo magnético.
    - El voltaje a través de la carga es menor que el de la fuente de entrada.
2. **Fase de apagado (Switch abierto):**
    - El interruptor se abre y la corriente deja de fluir directamente desde la fuente.
    - El inductor libera la energía almacenada, manteniendo el flujo de corriente hacia la carga a través del diodo.
    - Esto asegura que la carga reciba un suministro continuo de energía.

La velocidad a la que el interruptor se enciende y apaga (frecuencia de conmutación) y el tiempo que permanece encendido (ciclo de trabajo) determinan el nivel del voltaje de salida. Matemáticamente, el voltaje de salida (VoutV_{out}Vout) está relacionado con el voltaje de entrada (VinV_{in}Vin) y el ciclo de trabajo (DDD) mediante la fórmula:

Vout=Vin⋅DV_{out} = V_{in} \cdot DVout=Vin⋅D

Donde DDD es la relación entre el tiempo que el interruptor está encendido y el período total de conmutación.

- ## Ventajas de las fuentes step-down

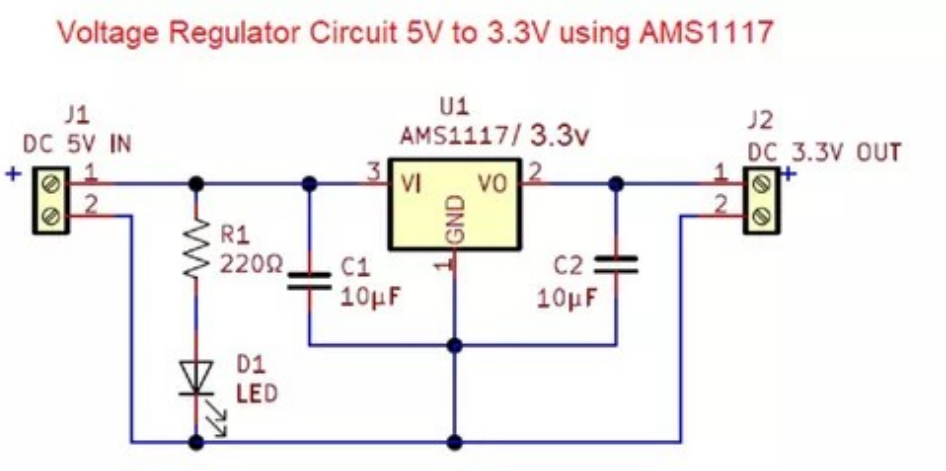
- Alta eficiencia (usualmente superior al 85%).
- Regulación precisa del voltaje de salida.
- Capacidad para manejar altas corrientes.

- **Aplicaciones comunes**

- Reducción de voltaje en sistemas electrónicos para alimentar microcontroladores, sensores, LEDs, etc.
- Cargadores de batería y adaptadores de corriente.
- Fuentes de alimentación reguladas para dispositivos móviles.

El enlace proporcionado explica también la diferencia entre los convertidores **step-down (buck)** y otros tipos como **step-up (boost)** y **buck-boost**, lo que ayuda a elegir el convertidor adecuado según la necesidad específica.

## *AMS1117*




- ***Características principales del AMS1117***

1. **Rango de voltaje de entrada**:
    - Funciona con un rango de entrada típicamente entre 4.5 V y 15 V, dependiendo de la versión específica.
2. **Voltajes de salida disponibles**:
    - Está disponible en versiones de salida fija (1.2 V, 1.5 V, 1.8 V, 2.5 V, 3.3 V y 5.0 V) y una versión ajustable.
3. **Capacidad de corriente**:
    - Puede proporcionar una corriente de salida de hasta 1 A, aunque necesita disipadores de calor adecuados si se opera a corrientes altas.
4. **Baja caída de tensión**:
    - La caída de tensión típica entre entrada y salida es de 1.1 V a plena carga (1 A), lo que significa que la tensión de entrada debe ser al menos 1.1 V mayor que la tensión de salida deseada.

## Pinout del AMS1117

1. **Vin** (Entrada de voltaje): Se conecta a la fuente de voltaje no regulada.
2. **Vout** (Salida de voltaje): Proporciona el voltaje regulado.
3. **GND**: Conexión a tierra común.

En la versión ajustable (AMS1117-ADJ), hay un pin adicional llamado **Adjust** que se utiliza para configurar el voltaje de salida.

## Cómo funciona internamente

El AMS1117 utiliza un diseño basado en un circuito de realimentación negativa. Internamente, consta de:

1. **Referencia de voltaje**:
    - Una referencia interna (normalmente de 1.25 V) asegura que el regulador mantenga un voltaje estable.
2. **Amplificador de error**:
    - Compara la referencia con una fracción del voltaje de salida (mediante un divisor resistivo interno o externo en la versión ajustable).
    - Si hay una desviación, el amplificador ajusta la conducción del transistor de paso (un MOSFET o BJT) para mantener el voltaje de salida constante.
3. **Transistor de paso**:
    - Actúa como una resistencia variable controlada, ajustando la corriente que pasa desde Vin hacia Vout.
4. **Protección interna**:
    - El AMS1117 incluye protecciones contra cortocircuitos y sobrecalentamiento, lo que lo hace robusto en diversas aplicaciones.

## • Aplicaciones típicas

1. **Estabilización de voltaje**:
    - Se utiliza para generar un voltaje constante (como 3.3 V o 5 V) a partir de una fuente no regulada, como baterías o fuentes de corriente continua.
2. **Conversión de niveles de voltaje**:
    - En circuitos donde ciertos componentes requieren niveles de voltaje específicos (como microcontroladores o sensores).
3. **Cargas de baja potencia**:
    - Ideal para alimentar circuitos que consumen corrientes bajas o moderadas.

## • Diseño del circuito con AMS1117

- Para garantizar un funcionamiento estable, se suelen usar capacitores de desacoplo en los pines de entrada y salida:
    - **Entrada (Vin)**: Se recomienda un capacitor de al menos 10 µF.
    - **Salida (Vout)**: Se recomienda un capacitor de 10 µF a 22 µF.

## • Ejemplo de conexión para AMS1117-3.3 (3.3 V fijo)

1. **Conexiones básicas**:
    - **Vin**: Conéctalo a una fuente de 5 V.
    - **GND**: Conéctalo a tierra común.
    - **Vout**: Proporciona 3.3 V regulados.
2. **Componentes adicionales**:
    - **Capacitor de entrada**: 10 µF (colocado cerca del regulador).
    - **Capacitor de salida**: 22 µF (para estabilizar el voltaje de salida).

- **AMS1117 Ajustable (AMS1117-ADJ)**
- Para obtener un voltaje de salida ajustable, puedes usar un divisor resistivo externo entre el pin de salida, el pin Adjust y tierra:
    - Vout=Vref⋅(1+R1R2)V_{\text{out}} = V_{\text{ref}} \cdot \left(1 + \frac{R1}{R2}\right)Vout=Vref⋅(1+R2R1), donde Vref=1.25 VV_{\text{ref}} = 1.25 \, VVref=1.25V.

Ejemplo:

- R1 = 2.2 kΩ y R2 = 1.1 kΩ proporcionan Vout=3.3 VV_{\text{out}} = 3.3 \, VVout=3.3V.

- **Ventajas y limitaciones**

## Ventajas:

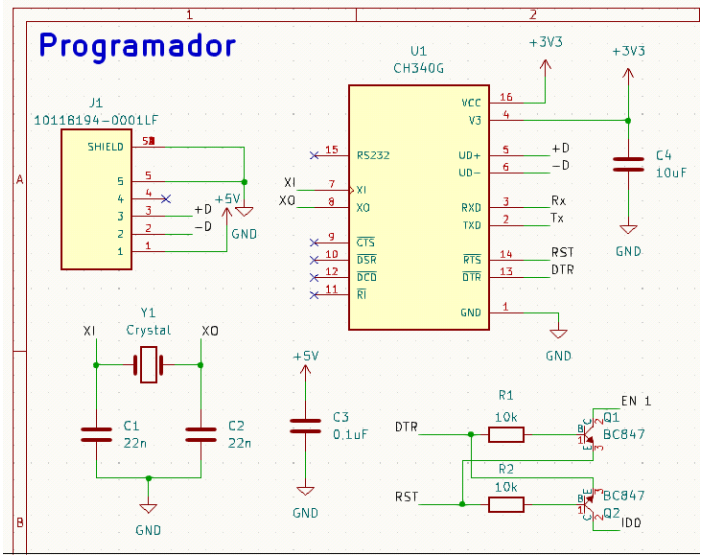
1. Fácil de usar y ampliamente disponible.
2. Bajo costo y alta fiabilidad.
3. Diseñado para bajo ruido en la salida.

## Limitaciones:

1. **Eficiencia baja**:
    - Al ser un regulador lineal, la energía se disipa como calor, especialmente cuando la diferencia entre Vin y Vout es alta.
2. **Requiere disipadores de calor**:
    - En aplicaciones de alta corriente, puede calentarse significativamente.

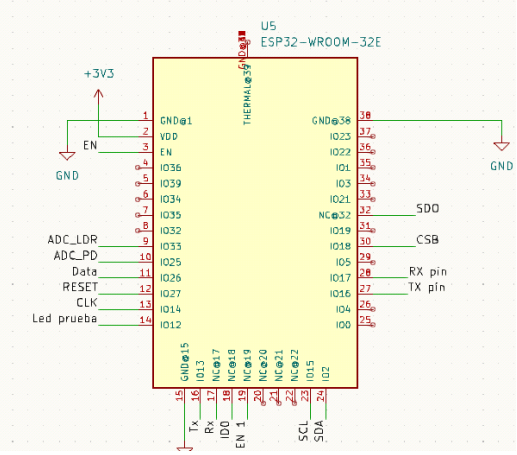
- # Programador:

El módulo CH340G es un adaptador USB a UART que permite conectar tu ordenador a microcontroladores como el ESP8266 o ESP32. Su función principal es facilitar la programación y comunicación con estos dispositivos.



## Componentes clave:

1. **Chip CH340G:** Convierte la señal USB a serie (UART).
2. **Pines UART (RX/TX):** Conectan al ESP (RX del módulo al TX del ESP y viceversa).
3. **Pines GPIO0 y RST:** Controlan el modo de programación y reinicio del ESP.
4. **Regulador de voltaje:** Convierte los 5V del USB a 3.3V para alimentar el ESP.
5. **Indicadores LED:** Muestran actividad de transmisión (TX) y recepción (RX).



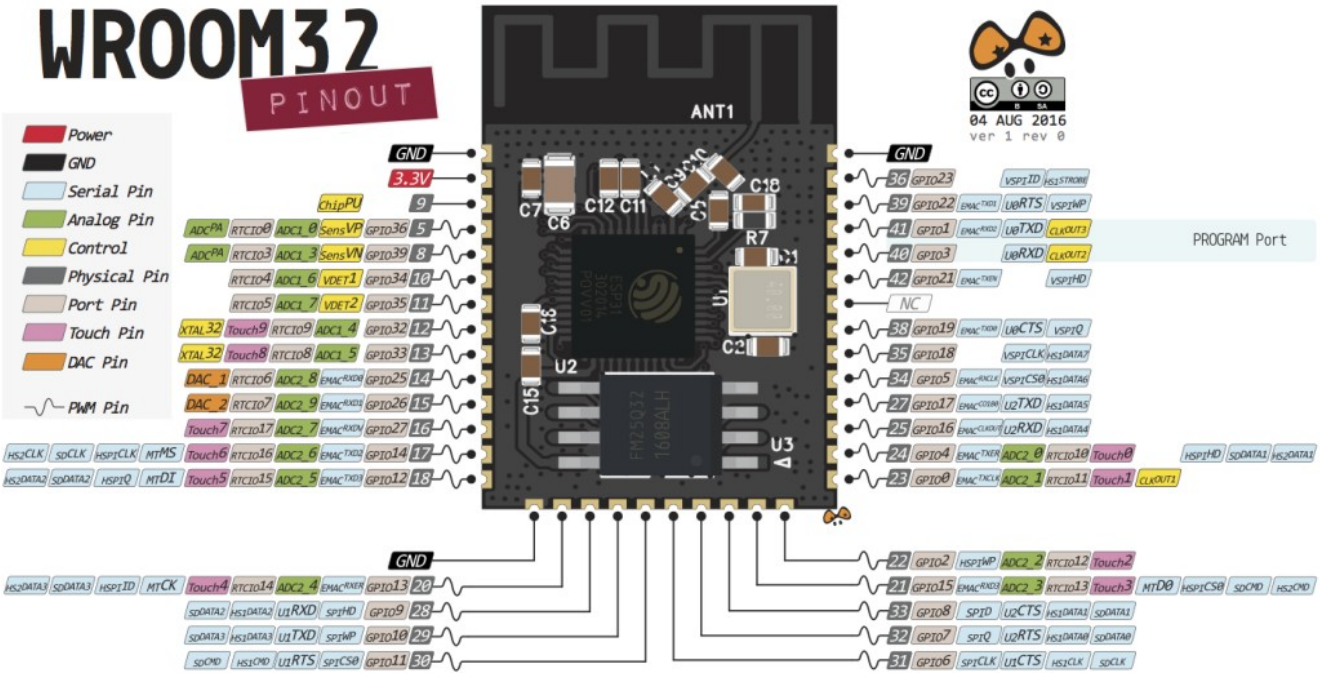

## Cómo usarlo:

1. Conecta el CH340G al ESP: VCC, GND, RX/TX, GPIO0, y RST.
2. Pon GPIO0 en LOW y presiona RST para entrar en modo programación.
3. Conecta el módulo al ordenador mediante USB.
4. Usa herramientas como Arduino IDE para cargar el código.
5. Vuelve GPIO0 a HIGH y reinicia para ejecutar el programa.

Es económico y muy útil para trabajar con ESP8266/ESP32. Si necesitas detalles sobre las conexiones o drivers, ¡dímelo!

- # Modulo/Microcontrolador

  El **ESP32** es un microcontrolador potente diseñado para proyectos IoT. Combina




Wi-Fi, Bluetooth y un procesador dual-core que trabaja hasta 240 MHz.

## Características principales:

1. **GPIOs versátiles**:
   - Entradas/salidas digitales.

- ADC (señales analógicas a digitales).
- DAC (señales digitales a analógicas).
- PWM (control de motores o LEDs).
- Pines táctiles y comunicación (I2C, SPI, UART).

2. **Conectividad**:
    - **Wi-Fi**: Compatible con redes 802.11 b/g/n.
    - **Bluetooth**: BLE y clásico para dispositivos.
3. **Energía**:
    - Funciona con 3.3V y soporta modos de bajo consumo para ahorrar batería.
4. **Usos avanzados**:
    - Incluye RTC (reloj interno), timers y periféricos integrados.

### Cómo funciona:

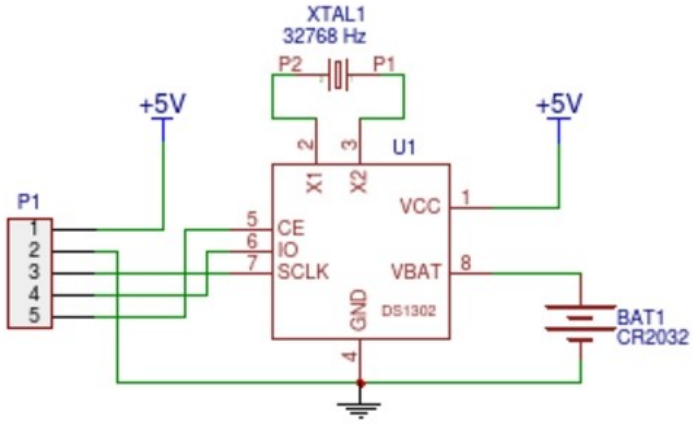
Programas el ESP32 (usando Arduino IDE, MicroPython, etc.) para ejecutar tareas como leer sensores, controlar actuadores y comunicarse por Wi-Fi o Bluetooth.

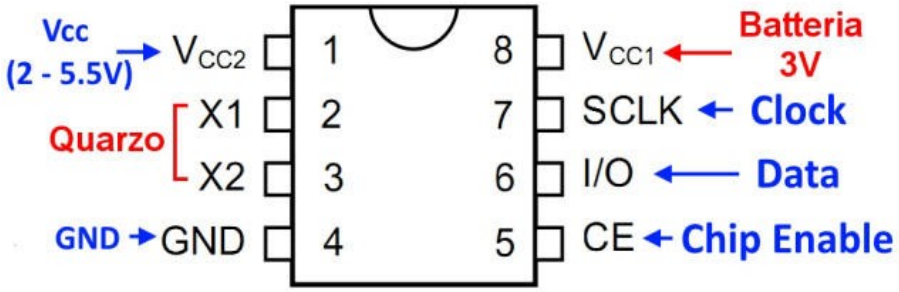
Ejemplo: Conectar un botón a un GPIO de entrada y un LED a uno de salida para encenderlo al presionar el botón.

- ## Modulo reloj:

### Módulo RTC DS1302: Cómo funciona

El módulo DS1302 es un reloj en tiempo real (RTC) que mantiene la hora y la fecha, incluso sin alimentación principal, gracias a una batería de respaldo




### Características principales

- **Reloj y calendario:** Segundos, minutos, horas, día, mes, año (ajuste automático para años bisiestos).
- **Formato:** 24 horas o 12 horas (AM/PM).
- **Respaldo:** Con batería (CR2032), funciona por años sin energía externa.
- **Memoria RAM:** 31 bytes para datos adicionales.
- **Comunicación:** Serial síncrona mediante 3 pines: **RST**, **CLK**, y **DAT**.

### Cómo funciona

1. **Sincronización:** El microcontrolador usa señales en **CLK** (reloj) y envía/recibe datos por **DAT**, activando el módulo con **RST**.
2. **Tiempo en BCD:** Los valores de tiempo se almacenan en formato binario codificado decimal para facilitar su uso.
3. **Autonomía:** Cuando no hay energía, cambia automáticamente al modo de bajo consumo usando la batería.

### Uso típico

- Conecta los pines al microcontrolador.
- Configura la hora una vez con un programa compatible.
- El DS1302 sigue contando el tiempo automáticamente.

### Ventajas

- Bajo consumo y fácil de usar.
- Mantiene el tiempo con precisión razonable.

---

## • Los sensores

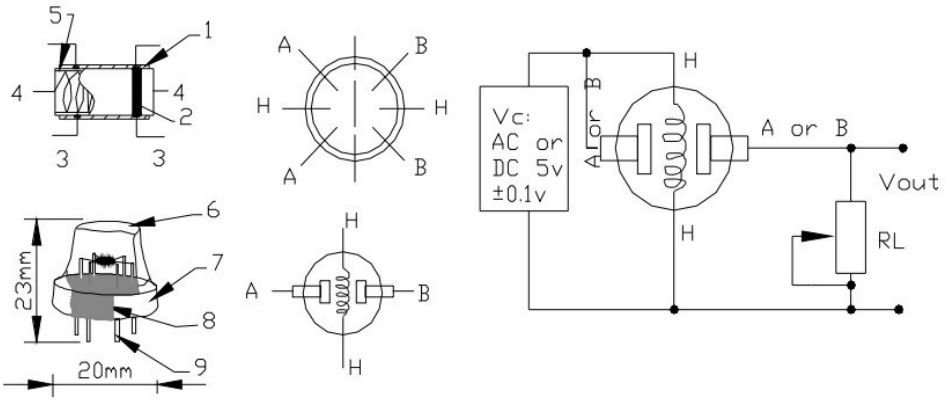
Estos son varios asi que los dividiremos en … grupos ya que cada uno es tan distinto entre si por lo que cada uno se merece su propio ítem detallado

## *1. Sensor de gas*

### Informe: Funcionamiento de los Sensores de Gas MQ

## Introducción

Los sensores de gas MQ son dispositivos electrónicos diseñados para detectar gases específicos en el ambiente. Son ideales para aplicaciones como sistemas de alarma, monitoreo ambiental y proyectos educativos. Cada modelo de la serie MQ está especializado en detectar diferentes gases.




## Estructura y Principio de Funcionamiento

1. **Estructura Interna**:
    - Los sensores MQ tienen un elemento sensor compuesto por óxido de estaño ($SnO_2$), un material semiconductor sensible a los gases.
    - Incluyen un calentador interno que estabiliza el funcionamiento del sensor.
2. **Principio de Detección**:
    - El óxido de estaño cambia su resistencia en presencia de ciertos gases.
    - Cuando un gas objetivo entra en contacto con el sensor, el material semiconductor adsorbe las moléculas del gas, lo que altera la resistencia eléctrica del sensor.
    - Esta variación de resistencia se convierte en una señal eléctrica proporcional a la concentración del gas.
3. **Conexión**:
    - Los sensores MQ generalmente tienen 4 pines:
        - **VCC**: alimentación.
        - **GND**: tierra.
        - **AO (Salida Analógica)**: proporciona un voltaje proporcional a la concentración de gas.
        - **DO (Salida Digital)**: salida binaria activada cuando se supera un umbral configurado.

## Modelos y Gases Detectados

1. **MQ2**:
    - Detecta gases inflamables como metano, propano y humo.
    - Usos: sistemas de detección de fugas de gas y alarmas de incendio.
2. **MQ3**:
    - Especializado en la detección de alcohol.
    - Usos: alcoholímetros y pruebas de etanol en el aire.
3. **MQ7**:
    - Diseñado para detectar monóxido de carbono (CO).
    - Usos: sistemas de seguridad en garajes y control de calidad del aire.

4. **MQ135**:
    - Detecta gases tóxicos y calidad del aire (amoníaco, dióxido de carbono, benceno).
    - Usos: monitoreo ambiental y sistemas de purificación de aire.

## Calibración

- Los sensores MQ requieren un período de "precalentamiento" para alcanzar un estado estable antes de la calibración.
- Se deben ajustar para obtener lecturas precisas de acuerdo con las condiciones del entorno.
- **Método de Calibración**:
    - Exponer el sensor a una concentración conocida del gas objetivo.
    - Ajustar los valores de salida para que coincidan con la concentración real.

## Ventajas y Limitaciones

- **Ventajas**:
    - Económicos.
    - Sensibles a una variedad de gases.
    - Fácil integración con microcontroladores como Arduino.
- **Limitaciones**:
    - No son selectivos, es decir, pueden reaccionar a varios gases al mismo tiempo.
    - Requieren un consumo energético moderado debido al calentador interno.
    - Necesitan recalibración periódica para mantener la precisión.

## Aplicaciones Prácticas

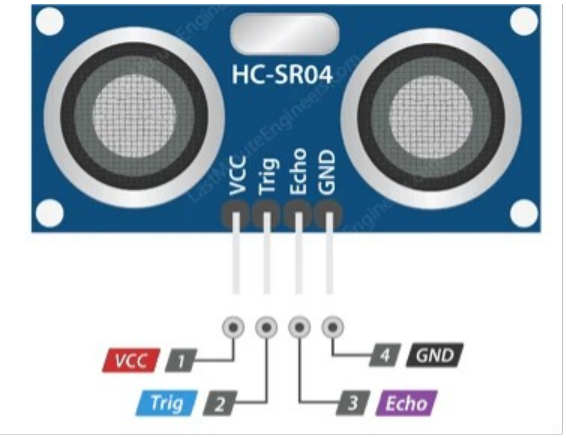
1. Alarmas de gas en hogares y edificios.
2. Sistemas de detección de humo e incendios.
3. Monitoreo ambiental en oficinas y fábricas.
4. Alcoholímetros para control vehicular.

## Conclusión

Los sensores MQ son una herramienta eficiente y accesible para detectar gases en diversos entornos. Sin embargo, su correcto funcionamiento depende de una calibración adecuada y un entendimiento de las limitaciones de cada modelo. Integrados con microcontroladores, permiten desarrollar sistemas de seguridad y monitoreo ambiental avanzados.

## *2. Sensor  de distancia ultrasonido*

El sensor HC-SR04 mide distancias utilizando ondas ultrasónicas. Su principio de funcionamiento se basa en emitir un pulso ultrasónico a través del transmisor y medir el tiempo que tarda en reflejarse en un objeto y regresar al receptor.



## Proceso de Medición:

1. Se envía un pulso ultrasónico desde el transmisor.
2. El pulso viaja hasta un objeto y se refleja de vuelta.
3. El sensor mide el tiempo transcurrido desde la emisión hasta la recepción del pulso.

## Cálculo de Distancia:

La distancia al objeto se calcula con la fórmula:

$$\text{Distancia} = \frac{\text{Tiempo} \times \text{Velocidad del sonido}}{2}$$

- **Velocidad del sonido**: Aproximadamente 343 m/s (o 0.034 cm/µs) en aire a 20°C.
- El factor 2 se debe a que el tiempo medido incluye el trayecto de ida y vuelta del pulso.

Por ejemplo: Si el tiempo medido es 500 µs, la distancia será:

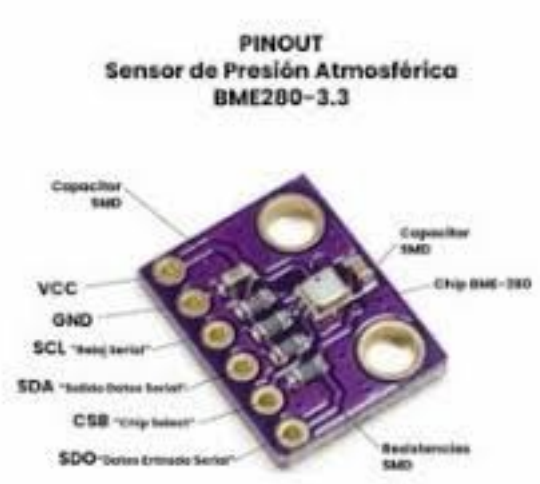
$$\text{Distancia} = \frac{500\,\mu s \times 0.034\,\text{cm/µs}}{2} = 8.5\,\text{cm}$$

## Limitaciones:

- Rango efectivo: Entre 2 cm y 400 cm.
- Sensibilidad: Afectada por la temperatura, humedad y la superficie del objeto.

### *3. Sensor de presión atmosférica*

El módulo sensor de presión barométrica BMP180 es un dispositivo compacto diseñado para medir la presión atmosférica, la temperatura y calcular la altitud con gran precisión. Este sensor, fabricado por Bosch, utiliza tecnología piezo-resistiva, que le otorga alta precisión, bajo consumo de energía y estabilidad a largo plazo, ideal para aplicaciones como drones, estaciones meteorológicas y sistemas de navegación.




## Funcionamiento del BMP180

1. **Medición de presión y altitud**: El sensor mide la presión atmosférica en un rango de 300 a 1100 hPa, con una resolución de 1 Pa y una precisión de hasta 0.03 hPa. A partir de la presión, se puede calcular la altitud relativa al nivel del mar mediante ecuaciones estándar.

2. **Medición de temperatura**: Incluye un sensor interno que mide la temperatura ambiente con una precisión de 1 °C y una resolución de 0.1 °C. Esta medición es crucial para compensar las lecturas de presión y aumentar su precisión.
3. **Comunicación**: Se conecta a microcontroladores (como Arduino) mediante la interfaz I2C, utilizando solo dos líneas de datos. Esto facilita la integración en proyectos electrónicos.
4. **Aplicaciones**: Además de calcular altitudes, se usa en drones para el control de vuelo, en sistemas de navegación para medir cambios en la presión y en IoT para monitoreo ambiental.

## Características técnicas principales

- **Voltaje de operación**: 3.3V a 5V DC.
- **Rango de presión**: 300-1100 hPa.
- **Rango de altitud medible**: 0-9100 metros.
- **Frecuencia de muestreo**: Hasta 120 Hz.
- **Consumo de energía**: Muy bajo, ideal para sistemas autónomos.

# *4. Sensor de temperatura y humedad*

El **DHT11** y el **DHT22** son sensores digitales que permiten medir temperatura y humedad en un entorno. Ambos son fáciles de usar, económicos y ampliamente utilizados en proyectos de electrónica, especialmente en aplicaciones con microcontroladores como Arduino y Raspberry Pi. Aunque tienen funcionalidades similares, existen diferencias clave en sus características técnicas.


## Características Comunes

1. **Salida digital**: La información de temperatura y humedad se transmite mediante un solo pin de datos en formato digital, lo que facilita su lectura.
2. **Integración sencilla**: Son compatibles con la mayoría de los microcontroladores y placas de desarrollo.
3. **Alimentación**: Operan típicamente con un rango de voltaje de 3.3V a 5V, lo que los hace versátiles.
4. **Precisión aceptable**: Aunque no son sensores de alta precisión, ofrecen valores adecuados para aplicaciones domésticas o educativas.

## DHT11

- **Rango de temperatura**: 0°C a 50°C, con una precisión de ±2°C.
- **Rango de humedad**: 20% a 90%, con una precisión de ±5%.
- **Tiempo de respuesta**: Aproximadamente 1 segundo.
- **Costo**: Es más económico en comparación con el DHT22.
- **Limitación**: Menor rango y precisión en comparación con el DHT22.

## Aplicaciones Comunes

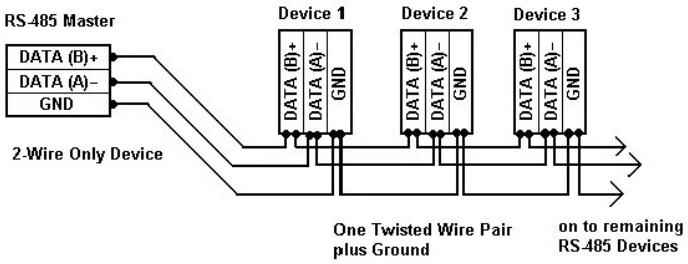
- Monitoreo de condiciones ambientales.
- Automatización del hogar (como control de clima).
- Proyectos educativos de electrónica.

- Control en invernaderos o espacios cerrados.

Para hacer el paso de una placa a otra vamos a hablar de el modulo que conecta estas 2 placas y su información

# • Rs485

El estándar RS-485 (también conocido como TIA/EIA-485) es una especificación de comunicación en serie ampliamente utilizada en aplicaciones industriales y de automatización. Su diseño permite la transmisión de datos en entornos difíciles y sobre largas distancias, ofreciendo robustez y eficiencia. A continuación, se presentan los aspectos más relevantes del RS-485:



2-Wire RS-485 Connections

## 1. Introducción al RS-485

RS-485 es un estándar definido por la TIA (Telecommunications Industry Association) y la EIA (Electronic Industries Alliance). Proporciona una interfaz eléctrica para sistemas de comunicación multipunto, siendo ideal para aplicaciones donde varios dispositivos deben comunicarse a través de un solo par de cables.

## 2. Características Técnicas

- **Topología de red**:
  Permite configuraciones multipunto o multidrop, donde hasta 32 dispositivos pueden conectarse en un solo bus de datos.
- **Transmisión diferencial**:
  Utiliza señales diferenciales para minimizar interferencias electromagnéticas (EMI) y mejorar la inmunidad al ruido.
- **Velocidad y distancia**:
  - La velocidad de transmisión disminuye con el aumento de la distancia.
  - A 100 kbps, es posible alcanzar distancias de hasta 1.200 metros.
- **Voltajes de señal**:
  - La diferencia entre los niveles lógicos de la señal (líneas A y B) garantiza una comunicación robusta.
  - Los niveles lógicos estándar son:
    - Nivel lógico 1: A - B > +200 mV
    - Nivel lógico 0: A - B < -200 mV
- **Resistencia de terminación**:
  Para evitar reflexiones de señal, se recomienda colocar resistencias de terminación (generalmente de 120 ohmios) en ambos extremos del bus.

## 3. Ventajas del RS-485

- **Inmunidad al ruido**:
  Gracias a la transmisión diferencial, es adecuado para entornos industriales ruidosos.

- **Distancia de transmisión**:
  Permite la comunicación eficiente a largas distancias, superando las limitaciones de otros estándares como RS-232.
- **Capacidad multipunto**:
  Ideal para sistemas distribuidos donde varios dispositivos deben compartir el mismo bus.
- **Compatibilidad**:
  Soporta la integración con protocolos como Modbus y Profibus.

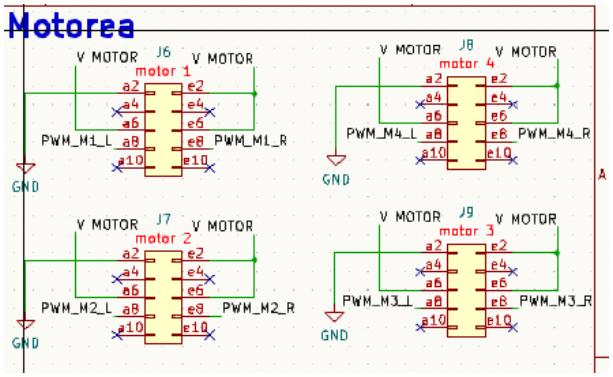
## 4. Aplicaciones del RS-485

- Sistemas de automatización industrial y control de procesos.
- Redes de sensores y actuadores en edificios inteligentes.
- Control de iluminación y sistemas HVAC.
- Comunicaciones en sistemas de transporte y energía (como SCADA).

## 5. Limitaciones y Consideraciones

- **Configuración y mantenimiento**:
  Requiere un diseño cuidadoso del bus, incluyendo terminaciones y control de polaridad, para evitar problemas de comunicación.
- **Número de dispositivos**:
  Aunque soporta 32 nodos por defecto, este límite puede extenderse utilizando repetidores o transceptores avanzados.
- **Dependencia de hardware**:
  La implementación puede ser más compleja comparada con protocolos punto a punto como RS-232.

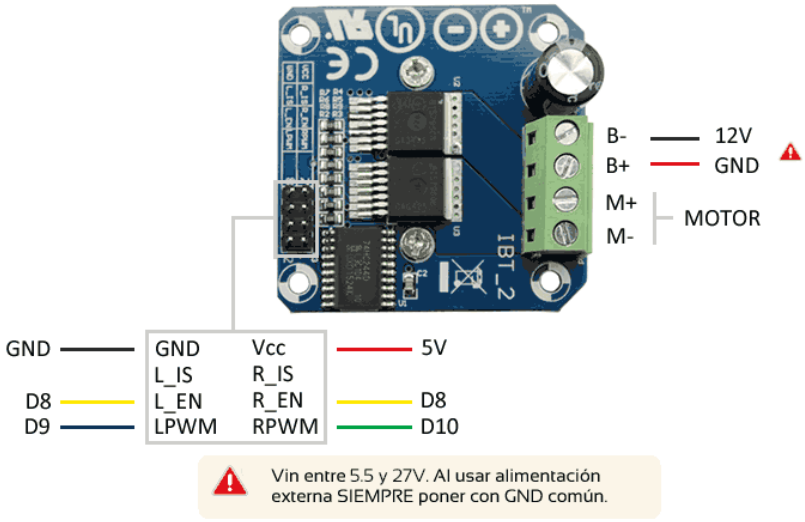
# • Placa de Movimiento con mega




• El segmento mas importante que encontramos en esta placa es el de los motores, esta parte de la placa tiene 4 conectores de tipo cable plano para conectarlo a los drivers BTS

El módulo **BTS7960**

ampliamente utilizado en proyectos con Arduino, es un controlador de motor diseñado para manejar motores de gran potencia. A continuación, te explico en detalle su funcionamiento y las características más relevantes:




## 1. Características principales

- **Capacidad de corriente**: Puede manejar hasta 43A continuos por canal, lo que lo hace ideal para motores de gran potencia.
- **Tensión de operación**: Generalmente soporta tensiones de hasta 27V, suficiente para la mayoría de motores DC.
- **Protección incorporada**: Incluye protecciones contra sobrecorriente, sobretemperatura y subtensión, lo que asegura una operación segura.
- **Tecnología H-Bridge**: Permite el control bidireccional del motor, es decir, puede girar en ambos sentidos.

## 2. Estructura del módulo

- **Chip BTS7960**: Es el corazón del módulo, basado en transistores MOSFET que manejan altas corrientes con baja disipación térmica.
- **Disipador de calor**: Ayuda a disipar el calor generado por el chip durante la operación prolongada.
- **Puertos de entrada y salida**:
    - **Entradas de control**: Pines que reciben señales PWM y de dirección desde un microcontrolador (como Arduino).
    - **Salidas de motor**: Conectores donde se conecta el motor.
    - **Entrada de alimentación**: Terminales para la alimentación externa del motor.

## 3. Funcionamiento básico

El módulo BTS7960 se controla mediante señales **PWM (Pulse Width Modulation)** y pines de dirección que determinan la velocidad y el sentido del giro del motor.

1. **Control de velocidad**:
    - La velocidad del motor se ajusta modificando el ciclo de trabajo de la señal PWM enviada al módulo.
    - A mayor ciclo de trabajo, mayor será el voltaje promedio aplicado al motor, aumentando su velocidad.
2. **Control de dirección**:
    - El módulo tiene dos pines de control de dirección: **R_EN (Right Enable)** y **L_EN (Left Enable)**.
    - Dependiendo de cuál esté activo, el motor girará en sentido horario o antihorario.
3. **Frenado y parada**:

- Si ambos pines de dirección están desactivados, el motor se detiene.
- Algunos módulos permiten frenar activamente, conectando ambos terminales del motor entre sí.

### 4. Conexión típica

1. **Alimentación externa**:
    - Se conecta una fuente externa de alta potencia para alimentar el motor (por ejemplo, una batería de 12V o 24V).
2. **Conexión al motor**:
    - Los terminales de salida del módulo se conectan directamente a los terminales del motor.
3. **Conexión al microcontrolador (Arduino)**:
    - Pines de control como **R_PWM**, **L_PWM**, **R_EN** y **L_EN** se conectan a pines digitales del Arduino.
    - Se usa un divisor de voltaje o nivel lógico si el módulo opera a tensiones distintas a las del microcontrolador.

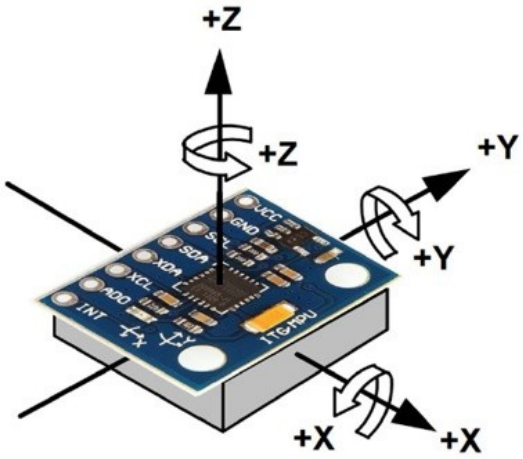
### 5. Ventajas del BTS7960

- Alta capacidad de corriente, ideal para motores grandes.
- Protección integrada, reduciendo el riesgo de daño.
- Operación sencilla con señales PWM.
- Compatible con la mayoría de microcontroladores.

### 6. Ejemplo de aplicación

Una posible aplicación es controlar un motor DC en un robot móvil. Usando el BTS7960, se puede ajustar la velocidad y dirección de las ruedas, permitiendo al robot moverse hacia adelante, atrás o girar, según los comandos del programa en el Arduino.

## • Giroscopio MPU6050




El módulo **MPU6050** es un sensor ampliamente utilizado en proyectos de electrónica y robótica. Combina un acelerómetro y un giroscopio en un solo chip, lo que lo hace ideal para medir movimientos, inclinaciones y velocidades angulares en aplicaciones como drones, robots y sistemas de estabilización. A continuación, te detallo las principales características y aspectos de este módulo basándome en la información general del sitio que compartiste:

### 1. Características principales

- **Chip principal:** MPU-6050 de InvenSense.
- **Acelerómetro:**
    - Mide aceleración lineal en los tres ejes: X, Y y Z.
    - Rangos ajustables: ±2g, ±4g, ±8g, ±16g.
- **Giroscopio:**

    - Mide velocidad angular en los tres ejes: X, Y y Z.
    - Rangos ajustables: ±250°/s, ±500°/s, ±1000°/s, ±2000°/s.
- **Resolución:**
    - Acelerómetro: 16 bits por eje.
    - Giroscopio: 16 bits por eje.
- **Comunicaciones:**
    - Interfaz I2C (dirección estándar: 0x68, configurable a 0x69).
- **Voltaje de operación:** 3.3V o 5V (la placa incluye regulador).
- **Frecuencia de muestreo:** Hasta 1 kHz.
- **Temperatura de operación:** -40°C a 85°C.
- **Sensor de temperatura integrado.**

## 2. Ventajas

- **Integración:** Combina acelerómetro y giroscopio en un solo módulo, reduciendo costos y espacio.
- **Configurabilidad:** Ajuste de rangos según la aplicación, permitiendo adaptarse a diferentes niveles de precisión y sensibilidad.
- **Facilidad de uso:** Compatible con microcontroladores comunes como Arduino y ESP32, gracias a bibliotecas preexistentes.
- **Precisión:** Resolución alta (16 bits) para obtener lecturas detalladas.
- **Compatibilidad:** Opera tanto con 3.3V como con 5V, gracias al regulador de voltaje integrado.

## 3. Componentes en el módulo físico

- **Chip MPU-6050:** Es el núcleo del módulo.
- **Regulador de voltaje:** Permite operar con voltajes de entrada de 3.3V o 5V.
- **Filtros de ruido:** Capacitores y resistencias para reducir interferencias en la señal.
- **Conector I2C:** Pines SDA (datos) y SCL (reloj), además de VCC y GND.

## 4. Aplicaciones comunes

- **Robótica:** Estimación de orientación y estabilización.
- **Drones y vehículos RC:** Control de vuelo y equilibrio.
- **Wearables:** Monitoreo de actividad física.
- **Gimbals:** Estabilización de cámaras.
- **Dispositivos médicos:** Análisis de movimientos.

## 5. Funcionamiento básico

- El acelerómetro mide la aceleración en las tres direcciones del espacio, lo cual incluye la fuerza de gravedad. Esto permite detectar inclinaciones o cambios en la posición.
- El giroscopio mide las velocidades angulares, permitiendo detectar movimientos rotacionales.
- Ambas mediciones se combinan a menudo mediante algoritmos como el **filtro de Kalman** o el **filtro complementario** para obtener una estimación precisa de la orientación en el espacio.

# • Modulo controlador de servos PCA9685:

## ¿Qué es el módulo PCA9685?

Es un controlador PWM de 16 canales basado en el chip PCA9685, diseñado para generar señales PWM con precisión y controlar dispositivos como servomotores, LEDs y otros periféricos. Permite manejar hasta 16 dispositivos simultáneamente con un solo microcontrolador, utilizando comunicación I2C.

## Características principales

1. **16 canales PWM independientes**: Cada canal puede emitir una señal PWM con diferentes configuraciones.
2. **Resolución de 12 bits**: Esto permite dividir cada ciclo de señal PWM en 4096 pasos, proporcionando una gran precisión.
3. **Frecuencia ajustable**: Entre 24 Hz y 1526 Hz.
4. **Dirección I2C configurable**: Se pueden conectar hasta 62 módulos en el mismo bus I2C (cada uno con una dirección única), controlando hasta 992 servos.
5. **Alimentación independiente**: Tiene un conector para alimentar los servos directamente, separado de la alimentación del PCA9685.

## ¿Cómo funciona?

# 1. Comunicación I2C

El PCA9685 se comunica con Arduino o cualquier otro microcontrolador a través del bus I2C (dos pines: SDA y SCL).

- **SDA**: Datos.
- **SCL**: Reloj. En el Arduino, estos pines suelen ser A4 y A5 para placas como Arduino Uno, o los pines dedicados para modelos como el Mega o Leonardo.

# 2. Generación de PWM

El PCA9685 genera una señal PWM de forma independiente para cada uno de sus 16 canales:

- Cada canal tiene dos registros de control: uno para establecer cuándo se enciende la señal y otro para definir cuándo se apaga.
- La resolución de 12 bits permite ajustar el ciclo de trabajo (duty cycle) con precisión. Por ejemplo, un valor de 0 a 4095 define la duración de cada pulso en un ciclo completo.

# 3. Frecuencia

La frecuencia de las señales PWM se establece de forma global para todos los canales. Esto se hace configurando el registro de frecuencia. Por ejemplo, para servos típicos, se usa una frecuencia de 50 Hz (20 ms por ciclo).

# 4. Control de servomotores

Los servomotores se controlan ajustando el ancho del pulso en la señal PWM:

- Un pulso corto (~1 ms) mueve el servo hacia un extremo.
- Un pulso largo (~2 ms) lo mueve hacia el otro extremo.
- Pulsos intermedios posicionan el servo en ángulos proporcionales.

Con el PCA9685, se configura la duración del pulso (en pasos de 12 bits) para cada canal que controle un servo.

## Conexiones típicas

1. **Alimentación del módulo**:
    - Conecta **VCC** (3.3V o 5V) y **GND** del PCA9685 a Arduino.
    - Conecta una fuente de alimentación externa al conector **V+** para los servos, ya que suelen requerir más corriente.
2. **Comunicación I2C**:

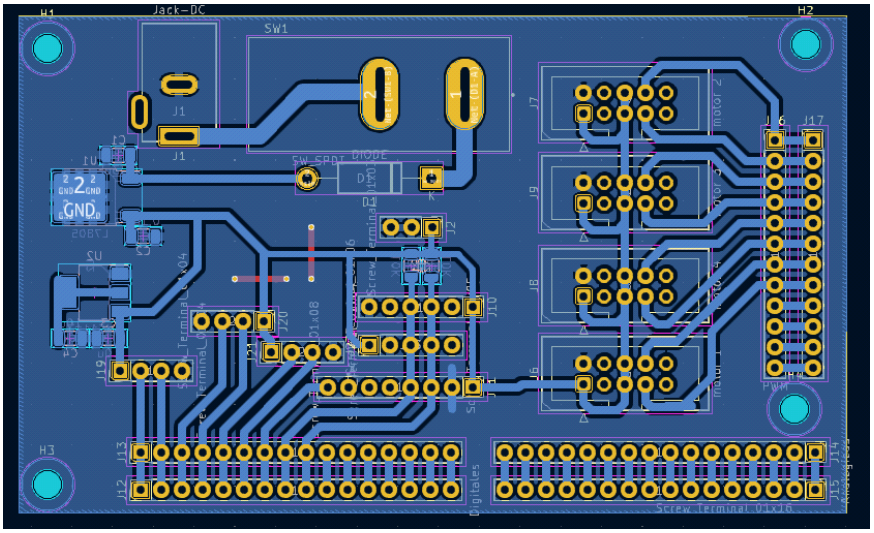
- Conecta **SCL** y **SDA** al Arduino.

3. **Salidas PWM**:
    - Los 16 pines de salida (del canal 0 al 15) se conectan a los pines de control de los servos.

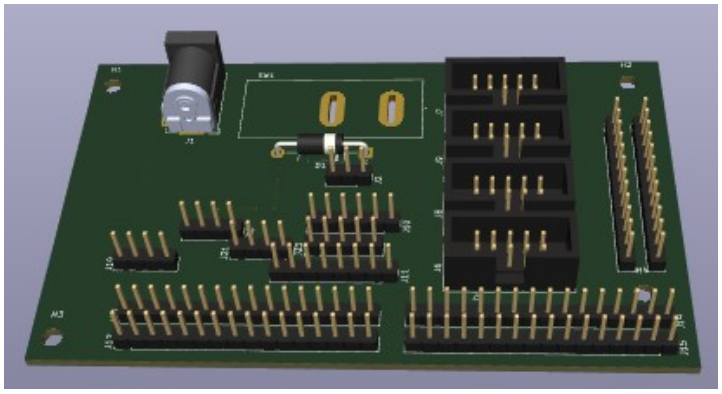
---

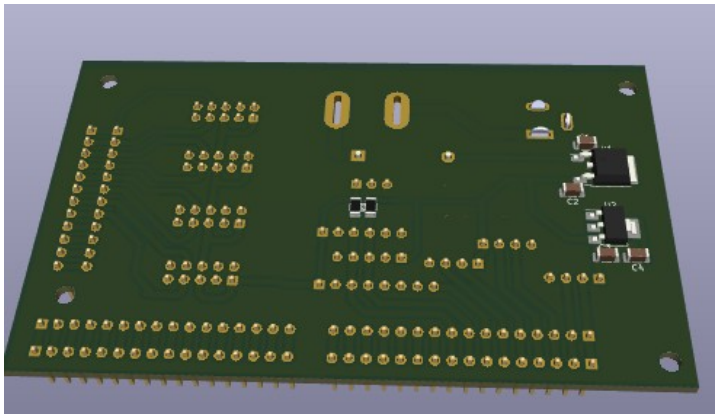
## • Pcb de las placas

### *Placa motores*

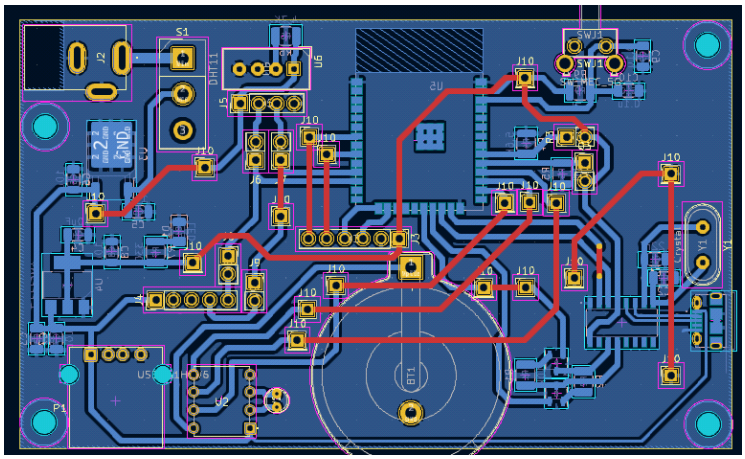



En esta placa tenemos lo mencionado anteriormente acomodado de tal forma que todo se conecte con todo y sea fácil la conexión y utilización de todos sus componentes.
Podemos apreciar los 4 conectores de drives de motor justo al lado de la pinera de PWM del lado totalmente opuesto tenemos la entrada y los reguladores. Ubicado en el centro inferior tenemos la pinera digital y la analógica, la digital podemos apreciar que se conecta a los sensores y módulos. Las analógicas no están conectadas a nada ya que todo lo que utilizamos en esta placa va a conectores digitales, pero optamos agregarlas igualmente ya que si en algún momento queremos utilizar un componente o modulo que use señales analógicas.

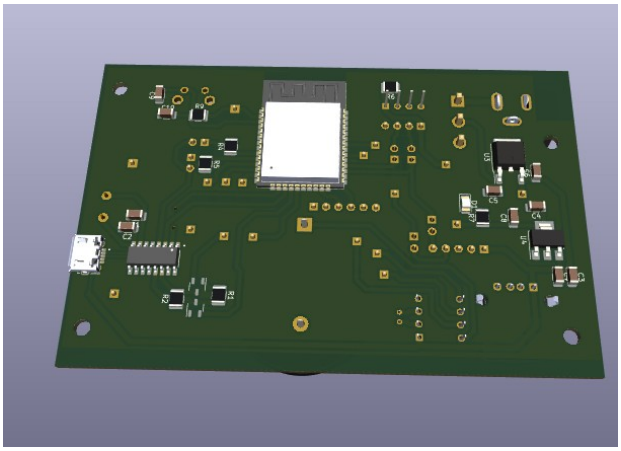


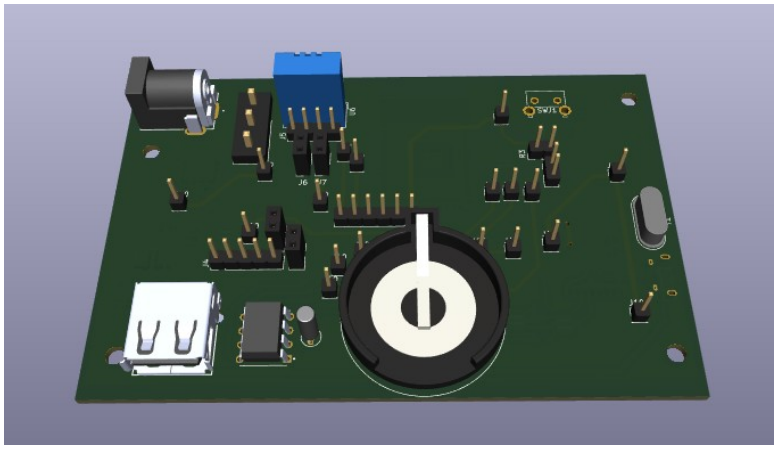


### *Placa sensores*



Esta placa es mas difícil de entender visualmente ya que tiene muchas cosas, podemos empezar por la esquina superior izquierda que tenemos la entrada el pulsador y el diodo, abajo de eso encontramos los reguladores de tensión de 5v y 3v3 para luego continuar con un usb que tiene una salida de 5v, justo a la derecha tenemos el reloj y la pila que mantiene en hora el robot. Por separado a todo esto tenemos en la esquina inferior izquierda el programador y la entrada micro usb donde le cargamos los datos del código.
Por ultimo tenemos arriba del medio a la derecha lo que es el microcontrolador y las conexiones extra como es el led de prueba o el botón reset.





---

## Software

Al igual que como hicimos con la placa voy a dividir esta explicacion para cada placa ya que cada una tiene su propio código,empezaremos con el codigo de control,este esta completamente hecho en arduino,usando el microcontrolador Arduino mega como mencionamos anteriormente
Este codigo lo voy a dividir según en que parte esta teniendo tres partes principales,el
Pre setup: donde declaramos la mayoria de variables y ponemos librerias
EL setup:donde ponemos las acciones que tienen que hacer los sensores y actuadores antes de empezar el constante movimiento
El loop:En este momento se le da una indicacion a cada sensor y actuador para que este funcionando en todo momento hasta que se le de una orden de fin

Este loop tendrá varias explicaciones para los distintos sensores que utilizen
Con esto
explicado ya podemos empezar con el

## Pre Setup

Empezamos poniendo las librerías

```
// Libraries
#include <SoftwareSerial.h>
#include "TinyGPS.h"  // https://www.arduino.cc/reference/en/libraries/tinygps/
#include <Wire.h>
#include "I2Cdev.h"    // https://github.com/jrowberg/i2cdevlib
#include "HMC5883L.h"  // https://github.com/jrowberg/i2cdevlib
//librerias para la Estabilizacion
#include "MPU6050.h"//pines 21 scl y 20 sda
#include <Servo.h>
//                                                    definicion de los servos
```

Estas librerías son para hacer funcionar algunos sensores(como el mpu6050 o el hmc5883l) o para hacer funcionar otros componentes como los servos o el conector de placas.

En esta parte tenemos la declaración de pines para los motores y los ultrasonicos

```
// Struct to store waypoints (GPS coordinates):
struct t_waypoint {
  float lat;
  float lon;
};
//_____________________________________________________
//cambiar las cordenadas por algunas de tu zona
// reach some place in my hometown! You can use

// {lat, lon}
t_waypoint nav_waypoints[] = {
  { -17.4176, -66.13265 },   // Point 1
  { -17.41757, -66.13282 },  // Point 2
  { -17.41773, -66.13282 },  // Point 3
  { -17.41776, -66.13265 }   // Point 4
};
```

Aquí tenemos las puntos a los que tendrá que ir al robot en su camino. Esto la hacemos con la ayuda de el gps y el compass antes mencionados.

Esto actualmente ya están puestos en el código y cada vez que llega a uno estos waypoints pasa al siguiente pero esta parte la vamos a ver con mas detalle próximamente en el apartados de control navigation

```
// Control signal pins for the the motor driver x4
int pwma1 = 2 , pwma2 = 3; //derecha adelante
int en_a1= 22 , en_a2 = 22;
int pwmb1 = 4 , pwmb2 = 5; //izquierda adelante
int en_b1= 23 , en_b2 = 23;
int pwmc1 = 6 , pwmc2 = 7; //derecha atras
int en_c1= 24 , en_c2 = 24;
int pwmd1 = 8 , pwmd2 = 9; //izquierda atras
int en_d1= 25 , en_d2 = 25;
int led=13;
//pines correctamente configurados pwm
//_____________________________________Ultrasonicos__________
int pud = 48;  // parlante derecha,trigger rigth
int pui = 49;  //configurado en el shield
int mud = 50;  // microfono derecha,echo right
int mui = 51;  //
```

Eso si bien no es todo lo que esta en esta parte del codigo,es la mas importante,con esto ya explicado podemos pasar al

## Setup

```
void setup() {
  // Initialize control pins for the motor driver
  pinMode(pwma1, OUTPUT);  pinMode(pwma2, OUTPUT);
  pinMode(pwmb1, OUTPUT);  pinMode(pwmb2, OUTPUT);
  pinMode(pwmc1, OUTPUT);  pinMode(pwmc2, OUTPUT);
  pinMode(pwmd1, OUTPUT);  pinMode(pwmd2, OUTPUT);
  pinMode(en_a1, OUTPUT);  pinMode(en_a2, OUTPUT);
  pinMode(en_b1, OUTPUT);  pinMode(en_b2, OUTPUT);
  pinMode(en_c1, OUTPUT);  pinMode(en_c2, OUTPUT);
  pinMode(en_d1, OUTPUT);  pinMode(en_d2, OUTPUT);

  Stop();  // Stop the car

  Wire.begin();           // Initialize I2C comm. for the compass
  Serial.begin(9600);     // Initialize serial comm. for debugging
  serial_gps.begin(9600);  // Initialize serial comm. with the GPS module

  compass.initialize();  // Initialize the compass
  //Estabilizacion
  sensor.initialize();    //Iniciando el sensor
  //configuracion de los sensores
  servo_1.attach(2, 500, 2500);  servo_2.attach(4, 500, 2500);
  servo_3.attach(6, 500, 2500);  servo_4.attach(8, 500, 2500);//cambiar el Pin de cada uno(e
  //todos los servos en angulo 0
  servo_1.write(45); servo_2.write(45); servo_3.write(45); servo_4.write(45);
```

En este comenzamos poniendo a los motores como salidas por lo que no nos va dar datos si no que solo los van a recibir.

Iniciamos el puerto serial esto es solo para probar y coroborar algunas cosas como el funcionamiento de los sensores.

Tambien iniciamos el compass y el gps despues configuramos los servos y los ponemos para que esten a 45° de base con esto pasaremos al loop

## Loop

```
void loop() {
  bluetooth();
  Get_Compass_Heading();  // Read the digital compass

  if (Query_Gps()) {  // Query the GPS
    Gps_Dump(gps);    // Read the GPS

    // Store the new GPS reading in filter buffers
    Store_Gps_Reading(f_gps_reading_lat, f_gps_reading_lon);

    // Get filtered GPS latitude and longitude
    gps_filtered = Compute_Filtered_Gps();

    //movimiento de los servos conforme al suelo
    Estabilizacion();

    // Compute distance and heading to the goal
    Compute_Navigation_Vector(gps_filtered.lat, gps_filtered.lon);

    // Print data to the PC just for debugging
    Print_Data();
  }
  // Move the car to the goal
  Control_Navigation();
  //evita que el auto choque
  esquivar();
```

En el loop como tal no hay nada sino que llama a las acciones que vamos a hacer las cuales difinimos cada una por su parte,para hacerlo simple aquellos voids que llama son:
*Para hacer funcionar el bluetooth
*Para hacer que el gps haga lecturas
* Y llama a el void estabilizacion el cual se encargara de que ell robot este a 0° siempre
*Se fija cuanto falta hasta llegar a la meta
*Y controla el robot hasta llegar a la meta este esta en conjunto con el siguiente esquivar

Con esto estaria explicado,pero vamos a ver un poco mas en detalle estos voids o funciones
empezando con el

## Bluetooth

```
void bluetooth(){
    if (Serial.available())
  {
    char dato=Serial.read();
    Serial.print("Dato recibido: ");
    Serial.println(dato);
        switch(dato){
      case 'a':
        adelante(adelante_VEL);
        break;
      case 'r':
    //    Atras();
        break;
      case 'd':
        giro_derecha(SLOW_TURN_VEL);
        break;
      case 'i':
        giro_izquierda(FAST_TURN_VEL);
        break;
      case 'p':
        Stop();
        break;
    }
  }
}
```

Este es bastante simple,todo se basa en el hecho de que tenes un mando conectado a bluetooth,este al apretar una tecla,le manda a arduino un dato si el dato coincide con algunos de los del codigo (a,r,d,i,p) va a hacer algunas de las acciones que este en su respectiva linea de codigo

# Estabilización

Para la estabilización primero hay que hacer un código para eliminar o disminuir el ruido que tiene el mpu6050 ya que por la forma en la que esta armado este tiene un poco de ruido (esto no hace falta ponerlo en el código en general si no simplemente hacer un código aparte solo para esto y después de la calibración seguir) por lo que tambien deberiamos explicar este codigo.

De la misma forma que explicamos el anterior código por parte lo haremos con este empezando por el

Presetup

```
// Librerias I2C para controlar el mpu6050
// la libreria MPU6050.h necesita I2Cdev.h, I2Cdev.h necesita Wire.h
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"

// La dirección del MPU6050 puede ser 0x68 o 0x69, dependiendo
// del estado de AD0. Si no se especifica, 0x68 estará implicito
MPU6050 sensor;

// Valores RAW (sin procesar) del acelerometro y giroscopio en los ejes x,y,z
int ax, ay, az;
int gx, gy, gz;

//Variables usadas por el filtro pasa bajos
long f_ax,f_ay, f_az;
int p_ax, p_ay, p_az;
long f_gx,f_gy, f_gz;
int p_gx, p_gy, p_gz;
int counter=0;

//Valor de los offsets
int ax_o,ay_o,az_o;
int gx_o,gy_o,gz_o;
```

la mayor diferencia con el código importante explicado anteriormente es que en esta declaramos una mayor cantidad de variables,la razón de esto es que funcionan como filtro pasa bajos y también declaramos una menor cantidad de librerías ya que no tenemos tantos sensores

Setup

```
void setup() {
  Serial.begin(57600);   //Iniciando puerto serial
  Wire.begin();           //Iniciando I2C
  sensor.initialize();    //Iniciando el sensor

  if (sensor.testConnection()) Serial.println("Sensor iniciado correctamente");

  // Leer los offset los offsets anteriores
  ax_o=sensor.getXAccelOffset();
  ay_o=sensor.getYAccelOffset();
  az_o=sensor.getZAccelOffset();
  gx_o=sensor.getXGyroOffset();
  gy_o=sensor.getYGyroOffset();
  gz_o=sensor.getZGyroOffset();

  Serial.println("Offsets:");
  Serial.print(ax_o); Serial.print("\t");
  Serial.print(ay_o); Serial.print("\t");
  Serial.print(az_o); Serial.print("\t");
  Serial.print(gx_o); Serial.print("\t");
  Serial.print(gy_o); Serial.print("\t");
  Serial.print(gz_o); Serial.print("\t");
  Serial.println("nnEnvie cualquier caracter para empezar la calibracionnn");
  // Espera un caracter para empezar a calibrar
  while (true){if (Serial.available()) break;}
  Serial.println("Calibrando, no mover IMU");
```

or su parte el setup la diferencia es que al momento de iniciar te muestra los valores iniciales del offset pidiéndote que para inicial aprietes un botos eso seria todo lo destacable del setup

Loop

```
void loop() {
  // Leer las aceleraciones y velocidades angulares
  sensor.getAcceleration(&ax, &ay, &az);
  sensor.getRotation(&gx, &gy, &gz);

  // Filtrar las lecturas
  f_ax = f_ax-(f_ax>>5)+ax;
  p_ax = f_ax>>5;

  f_ay = f_ay-(f_ay>>5)+ay;
  p_ay = f_ay>>5;

  f_az = f_az-(f_az>>5)+az;
  p_az = f_az>>5;

  f_gx = f_gx-(f_gx>>3)+gx;
  p_gx = f_gx>>3;

  f_gy = f_gy-(f_gy>>3)+gy;
  p_gy = f_gy>>3;

  f_gz = f_gz-(f_gz>>3)+gz;
  p_gz = f_gz>>3;
```

```
  //Cada 100 lecturas corregir el offset
  if (counter==100){
    //Mostrar las lecturas separadas por un [tab]
    Serial.print("promedio:"); Serial.print("t");
    Serial.print(p_ax); Serial.print("\t");
    Serial.print(p_ay); Serial.print("\t");
    Serial.print(p_az); Serial.print("\t");
    Serial.print(p_gx); Serial.print("\t");
    Serial.print(p_gy); Serial.print("\t");
    Serial.println(p_gz);

    //Calibrar el acelerometro a 1g en el eje z (ajustar el offset)
    if (p_ax>0) ax_o--;
    else {ax_o++;}
    if (p_ay>0) ay_o--;
    else {ay_o++;}
    if (p_az-16384>0) az_o--;
    else {az_o++;}

    sensor.setXAccelOffset(ax_o);
    sensor.setYAccelOffset(ay_o);
    sensor.setZAccelOffset(az_o);

    //Calibrar el giroscopio a 0°/s en todos los ejes (ajustar el offset)
    if (p_gx>0) gx_o--;
    else {gx_o++;}
    if (p_gy>0) gy_o--;
    else {gy_o++;}
    if (p_gz>0) gz_o--;
    else {gz_o++;}

    sensor.setXGyroOffset(gx_o);
    sensor.setYGyroOffset(gy_o);
    sensor.setZGyroOffset(gz_o);

    counter=0;
  }
  counter++;
}
```

El loop esta dividido en 2 imágenes debido a la longitud del mismo,sin embargo la explicación no es tan amplia,en primer lugar conseguimos los valores de aceleración y rotación,a estos valores los hacemos pasar por un filtro pasa bajos

Esto hará que en el serial estemos viendo constantemente valores y cuando veamos valores cercanos a lo que debería ser de base 1g(para la aceleración) y 0° (para al rotación) debamos de parar manualmente el código lo ideal es que esta

parada sea por lo menos cada 100 valores dados

Una vez que utilizamos este código y disminuimos el ruido el mpu quedara con esa configuración en los valores base aunque cambiemos el código a futuro.

Con esto hecho ahora podemos pasar a la programación para la estabilización siendo de la siguiente manera

```
void MPU(){

  // Leer las aceleraciones y velocidades angulares
  sensor.getAcceleration(&ax, &ay, &az);
  sensor.getRotation(&gx, &gy, &gz);

  dt = (millis()-tiempo_prev)/1000.0;
  tiempo_prev=millis();

  //Calcular los ángulos con acelerometro
  float accel_ang_x=atan(ay/sqrt(pow(ax,2) + pow(az,2)))*(180.0/3.14);
  float accel_ang_y=atan(-ax/sqrt(pow(ay,2) + pow(az,2)))*(180.0/3.14);

  //Calcular angulo de rotación con giroscopio y filtro complemento
  ang_x = 0.98*(ang_x_prev+(gx/131)*dt) + 0.02*accel_ang_x;
  ang_y = 0.98*(ang_y_prev+(gy/131)*dt) + 0.02*accel_ang_y;
    Serial.println(ang_y_prev);
    Serial.println(gy);
  ang_x_prev=ang_x;
  ang_y_prev=ang_y;

  //Mostrar los angulos separadas por un [tab]

  Serial.print("Rotacion en X:  ");
  Serial.print(ang_x);
  Serial.print("tRotacion en Y: ");
  Serial.println(ang_y);

  delay(10);
}
```

El funcionamiento de esto es primero pedimos los valores al sensor mpu, después tenemos que realizar uno cálculos siendo estos bastante mas complicado si los comparamos con otros cálculos que se realizan como el de los uiltrasonicos ya que no simplemente es un valor que el sensor consigue y los transforma con u simple calculo(a diferencia del ultrasonico el cual lanza una onda y mide la distancia a través del tiempo de esa onda)si no que este utiliza un poco mas de física,ya que los tres ángulos están en conjunto el calculo para saber la inclinación también tendrá que serlo por lo que se utiliza la formula

$$AnguloX = \operatorname{atan}\left(\frac{y}{\sqrt{x^2+z^2}}\right)$$En caso de calcular el angulo y se intercambia de lugar la x por la y. En este calculo también se multiplica por 180/pi (usamos 3.14 en el código) este agregado es para convertir los radianes a grados

Con eso tendríamos los valores x e y,sin embargo este probablemente tenga muchos problemas,estos problemas son distintos si usamos el acelero metro o el giroscopio en nuestro caso(giroscopio) el problema que puede tener es que se acumule el drift con el tiempo, por lo que debemos agregar un filtro de complemento
para eso utilizamos el calculo

$$ángulo = 0.98\left(ángulo + \boldsymbol{\omega}_{giroscopio}\, dt\right) + 0.02(\boldsymbol{ang}_{acelerómetro})$$

donde Dt es el tiempo entre calculo y calculo y los valores 0.98 y 0.02 son valores que usamos como filtro siendo el del giroscopio pasa altos y el acelero metro por un pasa bajos. Estos valores no tiene que ser necesariamente esos sin embargo lo que debe de cumplir es el hecho de que actúen como su tipo de filtro y que la suma de ambos de 1

Con esto calculado podemos imprimir por pantalla la rotación en los ejes X e Y
Con esto terminamos la función

con esto quedaría la otra función llamada
Estabilizacion

```
void Estabilizacion(){
//-------------------------------------------------combinados-------
if(gx<=-5 && gy>=5){
servo_4.write(s4++);   servo_3.write(s3++);
servo_2.write(s2--);   servo_1.write(s1--);   Serial.println("Combinada 1"); }
if(gx>=5 && gy<=-5){
servo_4.write(s4--);   servo_3.write(s3--);
servo_2.write(s2++);   servo_1.write(s1++);   Serial.println("Combinada 2"); }

//-----------------------------------------------Base--------------------
else if(gx<=-5)
{ servo_4.write(s4++);   servo_3.write(s3++);   Serial.println("Aumentando el angulo de las patas laterales derechas hasta estabilizar"); }
else if(gx>=5)
{ servo_2.write(s2--);   servo_1.write(s1--);   Serial.println("disminuyendo el angulo de las patas laterales izquierda hasta estabilizar"); }
if(gy<=-5)
{ servo_1.write(s1++);   servo_3.write(s3++);   Serial.println("Aumentando el angulo de las patas traseras hasta estabilizar");
 } //No es gx lo que tiene que aumentar si no el angulo del servo
else if(gy>=5)
{ servo_2.write(s2--);   servo_4.write(s4--);   Serial.println("disminuyendo el angulo de las patas traseras hasta estabilizar");  }
}
```

Esta explicación es mas simple ya que simplemente se utiliza el mismo método todo el rato,primero pregunta que angulo tiene los ejes x e y y si alguno es mayor a 5 osea no esta estabilizado(ya sea positivo o negativo) significara que no esta estable y en caso de no estar estable hará algunas de las cuestiones de aquí vamos a ir 1 por 1 ahora
En la primera pregunta si -5 es mayor a X y si y es mayor a 5 es caso de cumplirse ambas condiciones va a significar que para hacer que este estable hay que mover las patas 3 y 4 para que su angulo disminuya(aumentar el angulo del servomotor hará que el angulo de las patas disminuya) y que el angulo de las patas 1 y 2 aumente tras esto imprime por pantalla combinada 1(esto es solo para corroborar que funcione correctamente)
En la segunda pregunta si -5 es mayor a y, y si x es mayor a 5 es caso de cumplirse ambas condiciones va a significar que para hacer que este estable hay que mover las patas 3 y 4 para que su angulo aumente y que el angulo de las patas 1 y 2 disminuya tras esto imprime por pantalla combinada 2
Eso es el caso de que ambas ejes estén en un angulo no estable, en caso de que alguna de los ejes este estable pero el otro no se harán las siguientes cuestiones
si -5 es mayor a x se aumentara el angulo de las patas laterales derechas(3 y 4) y se mostrara por pantalla ese mensaje
si x es mayor a 5 se disminuirá el angulo de las patas laterales izquierdas(2 y 1) y se mostrara por pantalla ese mensaje
si -5 es mayor a y se aumentara el angulo de las patas traseras(3 y 1) y se mostrara por

pantalla ese mensaje
si y es mayor a 5 se disminuirá el angulo de las patas delanteras(2 y 4) y se mostrara por pantalla ese mensaje

# Control Navigation

Esta seria la parte final la cual le indica hacia donde moverse al robot,si bien tenemos otra forma de hacer que se mueva la cual mencionamos antes que es a través del Bluetooth esta es puramente manual por lo que estarás constantemente dando ordenes al robot para que se mueva,este al contrario ya tendrá metas definidas las cuales mencionamos en el pre setup, de hacia donde se debe mover el robot como veremos a continuación pero primero hay que tener en cuenta que utiliza varias funciones distintas por lo que las iremos viendo 1 por 1 como veremos ahora

```
/* Computes filtered latitude and longitude using a moving average filter */
t_waypoint Compute_Filtered_Gps() {
  float lat_sum;
  float lon_sum;

  t_waypoint filtered_waypoint;

  lat_sum = 0;
  lon_sum = 0;

  // Add all values in each buffer
  for (int i = 0; i < buffer_fill_index; ++i) {
    lat_sum = lat_sum + buffer_gps_lat[i];
    lon_sum = lon_sum + buffer_gps_lon[i];
  }

  // Take the average
  filtered_waypoint.lat = lat_sum / float(buffer_fill_index);
  filtered_waypoint.lon = lon_sum / buffer_fill_index;

  return filtered_waypoint;  // Return filtered values
}
```

en la primera función se fija donde esta, los checkpoints marcados anteriormente

```
/* Gets a reading from the compass */
void Get_Compass_Heading(void) {
  // Obtain magnetic field components in x, y and z
  compass.getHeading(&mx, &my, &mz);

  // Calculate the X axis angle W.R.T. North
  compass_angle = atan2(my, mx);
  compass_angle = compass_angle * RAD_TO_DEG;
  compass_angle = compass_angle - mag_declination;

  // Always convert to positive angles
  if (compass_angle < 0) {
    compass_angle = compass_angle + 360;
  }
}
```

tras esto el compass nos asigna nuestros ejes para al momento de moverse hacia los checkpoints este no se pierda

```
void Compute_Navigation_Vector(float gps_lat, float gps_lon) {
  t_waypoint cur_waypoint;

  // Get current goal
  cur_waypoint = Get_Waypoint_With_Index(waypoint_index);

  float gps_f_lat_rad;
  float gps_f_lon_rad;
  float a_haversine = 0;
  float c_haversine = 0;
  float d_haversine = 0;
  float parcial = 0;
  float delta_lat = 0;
  float delta_lon = 0;

  // Compute the distance to the goal with Haversine formula
  //_____________________________________________Distancia hasta la meta_____
  delta_lat = radians(cur_waypoint.lat - gps_lat);
  gps_f_lat_rad = radians(gps_lat);
  waypoint_lat_rad = radians(cur_waypoint.lat);
  delta_lon = radians(cur_waypoint.lon - gps_lon);

  a_haversine = sin(delta_lat / 2.0) * sin(delta_lat / 2.0);
  parcial = cos(gps_f_lat_rad) * cos(waypoint_lat_rad);
  parcial = parcial * sin(delta_lon / 2.0) * sin(delta_lon / 2.0);
  a_haversine += parcial;

  c_haversine = 2 * atan2(sqrt(a_haversine), sqrt(1.0 - a_haversine));

  d_haversine = 6371000.0 * c_haversine;  // Multiply by Earth's radius in meters

  last_calc_dist = d_haversine;
```

```
// Check if we are inside the goal's tolerance radius
if (d_haversine < TOLERANCE_RADIUS) {
  Stop();           // Stop the car
  delay(3000);      // Delay just to check visually were exactly the car reached the goal
  Serial.println("se alcanzo la meta");
  digitalWrite(led,HIGH);
  waypoint_index++;  // Switch to the next waypoint
  delay(100);  // Beep to signal 'waypoint reached'
}

// Check if we reached all waypoints
if (waypoint_index == num_waypoints) {
  Stop();  // Stop the car
  delay(100);
  while (1)     ;  // Stop the program (reset Arduino board to repeat)
}
// Compute the adelante azimuth
// ************************************************
gps_f_lon_rad = radians(gps_lon);
waypoint_lon_rad = radians(cur_waypoint.lon);

waypoint_angle = atan2(sin(waypoint_lon_rad - gps_f_lon_rad) * cos(waypoint_lat_rad),
                       cos(gps_f_lat_rad) * sin(waypoint_lat_rad) - sin(gps_f_lat_rad) * cos(waypoint_lat_rad) * cos(waypoint_lon_rad - gps_f_lon_rad));

waypoint_angle = waypoint_angle * 180 / PI;  // Convert from radians to degrees

// Always convert to positive angles
if (waypoint_angle < 0)
{   waypoint_angle += 360; }
}
//                                                  Camino a la Meta
```

en la siguiente función se va preguntando constantemente cuanto falta para llegar a la meta,si la llegamos a la meta y si ya alcanzamos todas las metas a través de la formula de harversine

```
void Control_Navigation() {
  heading_error = (waypoint_angle - compass_angle);  // Compute the heading error

  // Correct angle for wrap around
  if (heading_error < -180) {
    heading_error = heading_error + 360;
  }
  if (heading_error > 180) {
    heading_error = heading_error - 360;
  }

  // ----- Stepped proportional control
  // The error is between +5 and +45 (for ANGLE_RANGE_DIV = 0.25):
  if (heading_error > MIN_HEADING_ANGLE && heading_error <= MAX_HEADING_ANGLE * ANGLE_RANGE_DIV) {
    // Turn right
    giro_derecha(SLOW_TURN_VEL);
    str_action = "Right";
  }

  // The error is between +45 and +180 (for ANGLE_RANGE_DIV = 0.25):
  else if (heading_error > MAX_HEADING_ANGLE * ANGLE_RANGE_DIV && heading_error <= MAX_HEADING_ANGLE) {
    // Turn right fast
    giro_rapido_derecha(FAST_TURN_VEL);
    str_action = "Right fast";
  }
  // The error is between -5 and -45 (for ANGLE_RANGE_DIV = 0.25):
  else if (heading_error < -MIN_HEADING_ANGLE && heading_error >= -MAX_HEADING_ANGLE * ANGLE_RANGE_DIV) {
    // Turn left
    giro_izquierda(SLOW_TURN_VEL);
    str_action = "Left";
  }

  // The error is between -45 and -180 (for ANGLE_RANGE_DIV = 0.25):
  else if (heading_error < -MAX_HEADING_ANGLE * ANGLE_RANGE_DIV && heading_error >= -MAX_HEADING_ANGLE) {
    // Turn left fast
    giro_rapido_izquierda(FAST_TURN_VEL);
    str_action = "Left fast";
  }
  // The error is between -5 and +5 (for ANGLE_RANGE_DIV = 0.25):
  else if (heading_error >= -MIN_HEADING_ANGLE && heading_error <= MIN_HEADING_ANGLE) {
    // adelante
    adelante(adelante_VEL);
    str_action = "adelante";
  }
  // Just for debugging:
  else {
    str_action = "Not defined";
  }
}
```

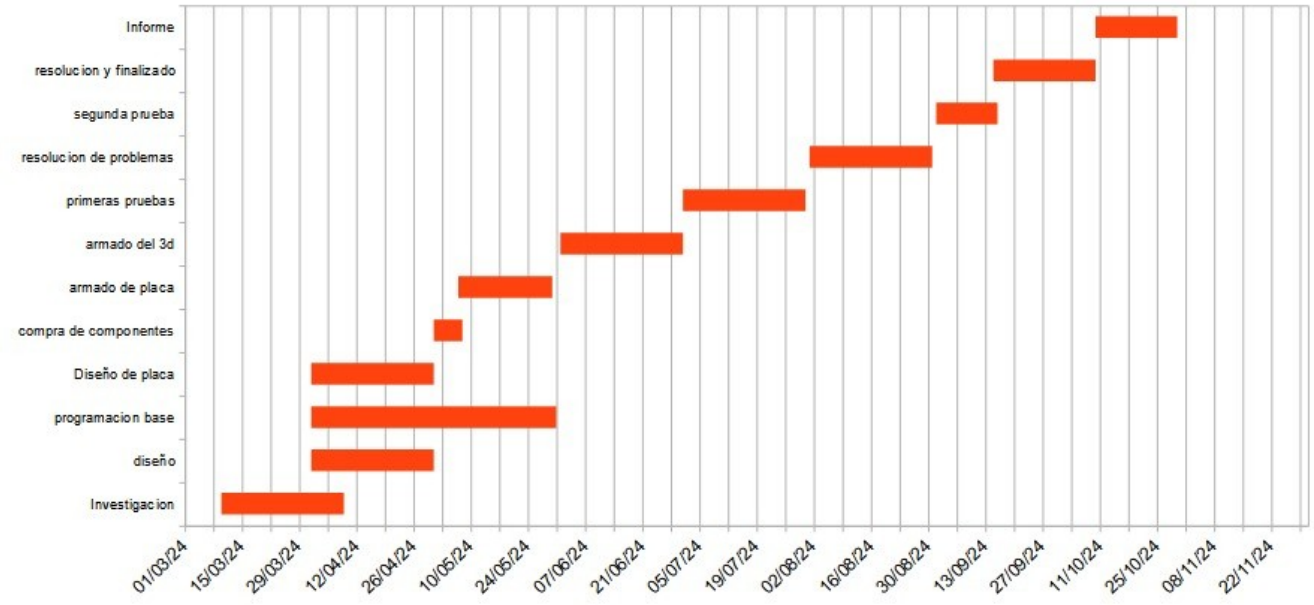
por ultimo en la funcion final es la que da el movimiento para que el robot se este moviendose correctamente hacia la meta

Tambien por ultimo esta la funcion esquivar la cual hace que el robot se mueva si uno de los ultrasonicos detecta algo a 20 cm de distancia este se movera a la izquierda si detecta a la derecha y se movera a la derecha si detecta algo a la izquierda

```
void esquivar(){
  if(di<<20 || dc<<20)
  {  giro_rapido_derecha(FAST_TURN_VEL);    delay(20);  }
  if(dd<<20)
  {  giro_rapido_izquierda(FAST_TURN_VEL);    delay(20); }
}
```

---

## Diagrama de gantt




---

## Presupuesto total y parcial del proyecto

| Producto | precio | cantidad | total | fuente | | Precio final del proyecto |
|---|---|---|---|---|---|---|
| Placa 30x30 simple fax | $35.222,00 | 1 | $35.222,00 | Mercado libre | | $727.497,00 |
| Pineras macho x40 | $4.000,00 | 4 | $16.000,00 | Mercado libre | | |
| Pineras hembra x40 | $4.319,00 | 4 | $17.276,00 | Mercado libre | | |
| Esp32 | $15.000,00 | 1 | $15.000,00 | Mercado libre | | |
| Arduino mega | $37.900,00 | 1 | $37.900,00 | Mercado libre | | |
| Servomotores | $12.100,00 | 4 | $48.400,00 | Mercado libre | | |
| Controladora de servos | $11.938,00 | 1 | $11.938,00 | Mercado libre | | |
| Giroscopio MPU6050 | $12.007,00 | 1 | $12.007,00 | Mercado libre | | |
| Programador | $5.077,00 | 1 | $5.077,00 | Mercado libre | | |
| Drivers BTS | $23.147,00 | 4 | $92.588,00 | Mercado libre | | |
| Rs485 | $3.150,00 | 2 | $6.300,00 | Mercado libre | | |
| Sensores ultrasonido | $3.100,00 | 3 | $9.300,00 | Mercado libre | | |
| Sensor de gas | $5.500,00 | 1 | $5.500,00 | Mercado libre | | |
| Fuente step down | $3.869,00 | 2 | $7.738,00 | Mercado libre | | |
| Sensor de luz ultra violeta | $19.165,00 | 1 | $19.165,00 | Mercado libre | | |
| Sensor de presión ad | $6.693,00 | 1 | $6.693,00 | Mercado libre | | |
| Ams1117 | $2.244,00 | 2 | $4.488,00 | Mercado libre | | |
| Lm7805 | $2.772,00 | 2 | $5.544,00 | Mercado libre | | |
| Resistencias smd | $1.850,00 | 3 | $5.550,00 | Mercado libre | | |
| Capacitores smd | $4.843,00 | 1 | $4.843,00 | Mercado libre | | |
| estaño | $10.900,00 | 1 | $10.900,00 | Mercado libre | | |
| Batería | $219.389,00 | 1 | $219.389,00 | Mercado libre | | |
| Chapa para el chasis | $38.400,00 | 1 | $38.400,00 | Mercado libre | | |
| Filamento de PLA | $43.285,00 | 1 | $43.285,00 | Mercado libre | | |
| ESP Cam | $23.994,00 | 1 | $23.994,00 | Mercado libre | | |
| tornillos/ resortes / rulemanes | $25.000,00 | 1 | $25.000,00 | Bulonera coiro | | |

i