Dispositivo de medición múltiple

Málaga, 10 de Junio de 2024

Fdo.: Nassim El Ouardighi

INDICE

Orden	Concepto	Páginas

1. MEMORIA

1.1. Memoria Descriptiva

1.1.1. Objeto

Medidor de Distancia y Ángulos con Arduino

El objetivo de este proyecto es construir un dispositivo capaz de medir distancias, ángulos, niveles y revoluciones por minuto (RPM) utilizando una variedad de sensores y módulos conectados a un microcontrolador Arduino. El proceso implica seleccionar sensores adecuados, ensamblar y programar los componentes electrónicos, y diseñar una carcasa impresa en 3D para alojar todos los componentes.

1.1.2. Antecedentes

Este proyecto se basa en la combinación de tecnologías existentes para crear un dispositivo multifuncional. Existen varios dispositivos comerciales que realizan tareas similares, como telémetros láser y niveles electrónicos. Sin embargo, este proyecto busca integrar múltiples funciones en un solo dispositivo personalizado, proporcionando tanto la capacidad de medición de larga y corta distancia como la medición de ángulos y niveles.ç

1.1.3. Justificación

Para medir distancias con precisión en diferentes rangos, se utilizarán dos tipos de sensores:

- **SHARP GP2Y0A21**: Este sensor infrarrojo mide distancias de hasta 80 cm con una resolución de centímetros, ideal para rangos más largos.
- **VL53L0X**: Un sensor de tiempo de vuelo basado en láser que mide distancias de hasta 20 cm con precisión milimétrica, adecuado para mediciones más precisas a corto alcance.

Para la medición de ángulos y detección de niveles, se utilizará el módulo giroscópico y acelerómetro **MPU6050**, que permite medir ángulos y realizar diversas operaciones de cálculo.

Para medir la longitud, se utilizará un módulo codificador giratorio acoplado a una rueda impresa en 3D. Conociendo el perímetro de la rueda, se puede calcular la distancia recorrida al girar.

Para medir las RPM, se empleará un módulo de infrarrojos con un LED y un fototransistor, contando las rotaciones mediante una franja blanca en el eje giratorio.

1.1.4. Datos de partida

Los datos iniciales necesarios para desarrollar el proyecto incluyen:

- **Sensores**: SHARP GP2Y0A21, VL53L0X.
- **Módulos**: MPU6050, codificador giratorio, módulo de infrarrojos, puntero láser.
- **Microcontrolador**: Arduino.
- **Componentes adicionales**: Pulsadores, batería de 3.7V, módulo de carga con conector USB, pantalla OLED, interruptor deslizante.

1.1.5. Análisis y Descripción del circuito

Vamos a proceder a describir el circuito por partes:

1. **Medición de Distancia**:
    - **Sensor SHARP GP2Y0A21**: Conectado al Arduino para medir distancias de hasta 80 cm. Este sensor proporciona una señal analógica proporcional a la distancia detectada.
    - **Sensor VL53L0X**: Utilizado para medir distancias de hasta 30 cm con alta precisión. Se conecta al Arduino mediante comunicación I2C.
2. **Medición de Ángulos y Niveles**:
    - **Módulo MPU6050**: Este módulo combina un giroscopio y un acelerómetro para medir ángulos y detectar niveles. Se comunica con el Arduino mediante I2C y se posicionará en el centro de la carcasa.
3. **Medición de Longitud**:
    - **Módulo Codificador Giratorio y Rueda Impresa en 3D**: Conociendo el perímetro de la rueda, el codificador giratorio conectado al Arduino permitirá calcular la distancia recorrida al girar.

4. **Medición de RPM**:
    - **Módulo de Infrarrojos**: Compuesto por un LED infrarrojo y un fototransistor, este módulo detecta las rotaciones de una franja blanca en el eje giratorio para medir las RPM.
5. **Interfaz de Usuario y Alimentación**:
    - **Botones sin contacto**: Utilizados para interactuar con el dispositivo mediante cambios en la capacitancia.
    - **Batería de 3.7V y Módulo de Carga**: Proporcionan la alimentación necesaria para el dispositivo, con un conector USB para recargar la batería.
    - **Pantalla OLED**: Muestra las mediciones y datos relevantes.
    - **Interruptor Deslizante**: Para encender y apagar el dispositivo.

Para completar el dispositivo, se diseñará y fabricará una carcasa impresa en 3D que albergue todos los componentes, asegurando su correcta disposición y protección.

2. PROGRAMACIÓN:

2.1 Descripción del código:

Subimos un arduino nano un programa que hemos desarrollado, este programa se encargará de comunicarse con los sensores, la pantalla oled y procesar los datos que recibe de los sensores para posteriormente mostrarlos en pantalla con las magnitudes correctas y de una forma legible para el usuario, además de esto también se encargará de cambiar el modo, encender el láser y establecer los valores a 0 para realizar nuevas mediciones.

A continuación explicaremos el código:

Librerías usadas en el código:

```
1   //////////////////////Librerias
2   #include <Wire.h>
3   #include <Adafruit_GFX.h>
4   #include <Adafruit_SH1106.h>
5   #include <VL53L0X.h>
6   //////////////////////
```

Librería Wire.h se utiliza para habilitar la comunicación I2C entre el arduino y otros dispositivos compatibles con I2C en este caso la pantalla OLED SH1106, el sensor VL53L0X y el MPU6050.

Librería Adafruit_GFX.h se utiliza para proporcionar una serie de primitivas gráficas (líneas, rectángulos, círculos, texto, etc.) que permiten a los desarrolladores dibujar gráficos básicos y texto en estas pantallas de manera sencilla.

Librería Adafruit_SH1106.h sirve para trabajar con pantallas OLED que utilizan el controlador SH1106 y es necesaria para el funcionamiento de la pantalla.

Libreria VL53L0X.h es utilizada para interactuar con el sensor de distancia láser VL53L0X de STMicroelectronics en plataformas como Arduino. Este sensor de tiempo de vuelo (ToF) mide distancias de manera precisa usando un láser infrarrojo y puede medir distancias de hasta 2 metros. La librería proporciona una interfaz fácil de usar para configurar el sensor y leer las mediciones de distancia.

Asignar pines Entradas/Salidas

```
14  //INPUTS/OUTPUTS
15  #define zero_button 6
16  #define mode_button 7
17  #define laser_button 5
18  #define rot_data 12
19  //#define rot_clock 12
20  #define battery_in A1
21  #define sharp_in A0
22  #define IR_rpm_in 10
23  #define laser_out 9
```

Aquí definimos los pines que vamos a utilizar y a que corresponden

void setup:

La función `void setup()` es una parte fundamental de cualquier programa en Arduino. Es una de las dos funciones obligatorias (la otra es `void loop()`) y se utiliza para realizar configuraciones iniciales que sólo necesitan ejecutarse una vez al comienzo del programa.

```
79  void setup() {
80
81    analogReference(INTERNAL);
82
83    //Definir los pines como salidas o entradas y establecerlos como LOW o HIGH
84    pinMode(zero_button, INPUT);
85    pinMode(mode_button, INPUT);
86    pinMode(laser_button, INPUT);
87    pinMode(rot_data, INPUT_PULLUP);
88    //pinMode(rot_clock, INPUT_PULLUP);
89    pinMode(battery_in, INPUT);
90    pinMode(sharp_in, INPUT);
91    pinMode(IR_rpm_in, INPUT);
92    pinMode(laser_out, OUTPUT);
93    digitalWrite(laser_out,LOW);
```

La función analogReference() se utiliza para configurar la referencia de voltaje utilizada para las lecturas analógicas. En este caso, INTERNAL selecciona la referencia de voltaje interna del Arduino, que es de 1.1V en el caso de arduino Nano. Esto es útil cuando se necesita una referencia de voltaje estable y precisa para medir señales analógicas.
La función pinMode() se utiliza para configurar el modo de funcionamiento de los pines del Arduino, especificando si serán utilizados como entradas o salidas. El primer parámetro de la función representa el pin y el segundo es el

modo de funcionamiento del pin(INPUT, INPUT_PULLUP, INPUT_PULLDOWN o OUTPUT)

```
95  PCICR |= (1 << PCIE0);    //habilitamos las interrupciones por cambio de estado en el grupo de pines 8 a 15.
96  PCMSK0 |= (1 << PCINT4);  //Habilitamos la interrupción por cambio de pin específicamente para el pin 12.
97  //PCMSK0 |= (1 << PCINT3);  //Habilitamos la interrupción por cambio de pin específicamente para el pin 11.
98  PCMSK0 |= (1 << PCINT2);  //Habilitamos la interrupción por cambio de pin específicamente para el pin 10.
```

Las interrupciones por cambio de pin (Pin Change Interrupts) permiten que el microcontrolador responda a cambios en el estado de cualquier pin configurado para estas interrupciones. Esto es útil para responder rápidamente a cambios en el estado del encoder(encargado de medir la longitud) y el modulo IR(encargado de medir RPM) sin necesidad de verificar constantemente el estado de los pines en el bucle principal (loop()).
`PCICR` es el registro que habilita las interrupciones por cambio de pin para grupos específicos de pines.
La operación `PCICR|= (1 << PCIE0)` establece el bit `PCIE0` (el bit que habilita las interrupciones para el grupo de pines 8 a 15) en `PCICR` habilitando las interrupciones por cambio de estado para este grupo de pines.
La operación `PCMSK0|= (1 << PCINT4)` establece el bit `PCINT4` en `PCMSK0`, habilitando la interrupción por cambio de estado específicamente para el pin 12.
La operación `PCMSK0|= (1 << PCINT2)` establece el bit `PCINT2` en `PCMSK0`, habilitando la interrupción por cambio de estado específicamente para el pin 10.

```
100 display.begin(SH1106_SWITCHCAPVCC, 0x3C);
101 // inicializamos la pantalla OLED y establecemos la comunicacion I2C entre el arduino y la pantalla.
102 delay(100);
103 display.clearDisplay();
104 display.setTextSize(2);
105 display.setCursor(0, 30);
106 display.setTextColor(WHITE);
107 display.println("ELECTROPRO");
108 display.display();
109 delay(1000);
110 display.clearDisplay();
111 display.display();
```

En esta parte inicializamos la pantalla OLED y establecemos la comunicación I2C entre el arduino y la pantalla e imprimimos en la pantalla “ELECTROPRO” el nombre de nuestra empresa al pasar un segundo se limpia la pantalla

```
114   //VL53L0X sensor
115   sensor.init();
116   sensor.setTimeout(500);
117     // esta funcion establece cuanto tiempo dedica el sensor a realizar cada medicion.
118     // El tiempo es en microsegundos o sea son 200ms.
119   sensor.setMeasurementTimingBudget(200000);
120
121   //MPU6050 Giroscopio/Acelerometro
122   Wire.begin();                           //configuramos el arduino para actuar como maestro en el bus I2C
123   Wire.beginTransmission(0x68);           // Iniciamos la transmisión I2C con la dirección esclava 0x68
124   Wire.write(0x6B);                       //Seleccionamos el registro 0x6B (PWR_MGMT_1) del MPU6050
125   Wire.write(0x00);                       //Escribimos 0x00 en el registro 0x6B para desactivar el modo de reposo y activar el sensor
126   Wire.endTransmission(true);             //Finaliza la transmision I2C con el MPU6050
127   //Giroscopio
128   Wire.beginTransmission(0x68);           // Inicia la transmisión I2C con la dirección esclava 0x68
129   Wire.write(0x1B);                       // Selecciona el registro 0x1B (GYRO_CONFIG) del MPU6050
130   Wire.write(0x10);                       // Escribe 0x10 en el registro 0x1B para configurar el rango del giroscopio a ±1000°/s
131   Wire.endTransmission(true);             //Finaliza la transmision I2C con el giroscopio
132   //Acelerometro
133   Wire.beginTransmission(0x68);           // Inicia la transmisión I2C con la dirección esclava 0x68
134   Wire.write(0x1C);                       // Selecciona el registro 0x1C (ACCEL_CONFIG) del MPU6050
135   Wire.write(0x10);                       // Escribe 0x10 en el registro 0x1C para configurar el rango del acelerómetro a ±8g
136   Wire.endTransmission(true);             //  Finaliza la transmisión I2C
137   time = millis();
```

void loop

La función `void loop()` es una parte fundamental de cualquier programa en Arduino. Junto con la función `setup()`, forma la estructura básica de un sketch de Arduino. Mientras que `setup()` se ejecuta una sola vez al inicio del programa para realizar configuraciones iniciales, `loop()` se ejecuta continuamente después de `setup()`

```
142   void loop() {
143     analogReference(INTERNAL);
144     battery_level = map(analogRead(battery_in),0,1024, 0, 4730);
145     battery_level = battery_level / 1000;
```

Se realiza una lectura analógica del pin battery_level en milivoltios y luego convertirla en voltios dividiendo el valor anterior entre mil.

```
148   ////////////////////////////////CAMBIAR DE MODO////////////////////////////////
149   if(digitalRead(mode_button) && !mode_button_state)
150   {
151     data_state = digitalRead(rot_data);
152     mode_button_state = true;
153     Z_error = false;
154     Z_loop = 0;
155     IR_enable = false;
156     Gyr_rawZ_error = 0;
157     print_time = 0;  //Reinicia el tiempo de impresión para el modo ángulo(para el MODO 3 y el MODO 4)
158     mode = mode + 1;
159     if(mode > 5)
160     {
161       mode = 0;
162     }
163   }
```

Este fragmento de código es para cambiar de modo al pulsar el pulsador encargado de dicha tarea, lo que hace es sumarle 1 a la variable mode cada vez que pulsamos el pulsador, en total tenemos 6 modos, cada modo le corresponde un número del 0 al 5, en caso de que el valor de mode supere 5 su valor se restablece a 0.

```
171   ////////////////////////////////RESETEAR VALOR//////////////////////////////////
172   if(digitalRead(zero_button) && !zero_button_state)
173   {
174     counter = 0;
175     Z_error = false;
176     Total_angle_z = 0;
177     zero_button_state = true;
178   }
179   if(!digitalRead(zero_button) && zero_button_state)
180   {
181     zero_button_state = false;
182   }
```

Esta parte del código sirve para restablecer a 0 la medición para tomar una nueva al pulsar el pulsador zero (esto sirve sobre todo en la parte de medir la longitud con el encoder).

```
186   ////////////////////////////////LASER BUTTON//////////////////////////////////
187   if(digitalRead(laser_button) && !laser_button_state)
188   {
189     laser = !laser;
190     digitalWrite(laser_out, laser);
191     laser_button_state = true;
192
193   }
194   if(!digitalRead(laser_button) && laser_button_state)
195   {
196     laser_button_state = false;
197   }
```

Al pulsar el pulsador Laser se enciende el láser en caso de estar apagado previamente o se apaga en caso de estar encendido previamente

```
200   switch(mode){
201     case 0:  // Modo 0
202       MedirDistanciasLargas();
203       break;
204     case 1:  // Modo 1
205       MedirDistanciasCortas();
206       break;
207     case 2:  // Modo 2
208       MedirLongitud();
209       break;
210     case 3:  // Modo 3
211       MedirNivel();
212       break;
213     case 4:  // Modo 4
214       MedirAngulos();
215       break;
216     case 5:  // Modo 5
217       MedirRPM();
218       break;
219   }
```

Este fragmento de código se encarga de llamar a alguna de estas funciones (funciones encargadas de captar la información de los sensores/módulos procesarla y mostrarla en pantalla) según el valor de la variable mode que nosotros controlamos mediante el pulsador mode.

```
223   void DibujarCabecera(String modo){
224     display.clearDisplay();
225     display.setTextSize(1);
226     display.setTextColor(BLACK,WHITE);
227     display.setCursor(0,0);
228     display.print(modo);
229     display.setTextColor(WHITE);
230     display.setCursor(95,0);
231     display.print(battery_level);
232     display.print("V");
233   }
```

Esta función se encarga de limpiar la pantalla y de mostrar una cabecera en la pantalla que contiene dos datos, el modo en el que nos encontramos y el nivel de voltaje de la batería.

```
235   void FueraDeRango(){
236     display.setTextSize(2);
237     display.setCursor(0,10);
238     display.print("FUERA");
239     display.setCursor(0,30);
240     display.print("DE");
241     display.setCursor(0,50);
242     display.print("RANGO");
243     display.display();
244   }
```

Esta función muestra en pantalla el mensaje ‘FUERA DE RANGO’, esta función es llamada en el modo 0 MedirDistanciasLargas y modo 1 MedirDistanciasCortas al exceder el rango máximo del sensor.

```
246   void MedirDistanciasLargas(){
247     //analogReference(DEFAULT);
248     float Sharp_read = analogRead(sharp_in); // lectura del sensor Sharp
249     float Sharp_cm = pow(4727.4 / Sharp_read, 1.2134); // convertir en cm
250     int Sharp_cm_int = Sharp_cm;
251     float Sharp_inch = Sharp_cm * 0.393701;
252     if (Sharp_cm > 10 && Sharp_cm < 80)
253     {
254       DibujarCabecera("1_Distancia");
255       display.setTextSize(2);
256       display.setCursor(0,30);
257       display.print("cm: ");
258       display.print(Sharp_cm,0);
259       display.setCursor(0,50);
260       display.print("in: ");
261       display.print(Sharp_inch,0);
262       display.display();
263       delay(45);
264       Sharp_cm_previous = Sharp_cm_int;
265     }
266     if (Sharp_cm < 10)
267     {
268         DibujarCabecera("1_Distancia");
269       display.setTextSize(2);
270       display.setCursor(0,10);
271       display.print("DEMASIADO");
272       display.setCursor(0,30);
273       display.print("CERCA");
274       display.display();
275     }
276     if (Sharp_cm > 80)
277     {
278       DibujarCabecera("1_Distancia");
279       FueraDeRango();
280     }
281   }
```

Lee el valor analógico del pin `sharp_in` donde está conectado el sensor Sharp. El valor leído será entre 0 y 1023 y convierte la lectura del sensor en una distancia en centímetros usando una fórmula específica de calibración

derivada de las características del sensor almacena el valor en una variable para posteriormente usar dicha variable para obtener la medida en pulgadas. Con los datos obtenidos se mostrarán las medidas en pantalla tanto en centímetros como en pulgadas pero siempre que estén entre 10cm y 80cm porque ese es el rango del sensor, en caso de superar los 80cm se llamará a la función FueraDeRango() y si es inferior a 10cm se mostrará el mensaje “DEMASIADO CERCA”.

```
283  void MedirDistanciasCortas(){
284    float VL53L0X_mm = sensor.readRangeSingleMillimeters() - 40; //40mm es un error
285    float VL53L0X_cm = VL53L0X_mm / 10;
286    float VL53L0X_inch = VL53L0X_cm * 0.393701;
287    int VL53L0X_mm_int = VL53L0X_mm;
288
289    if (VL53L0X_mm < 300) // siempre que la medida no supere los 300mm
290    {
291      DibujarCabecera("2_Distancia");
292      display.setTextSize(2);
293      display.setCursor(0,10);
294      display.print("mm: ");
295      display.print(VL53L0X_mm,0);
296      display.setCursor(0,30);
297      display.print("cm: ");
298      display.print(VL53L0X_cm,1);
299      display.setCursor(0,50);
300      display.print("in: ");
301      display.print(VL53L0X_inch,1);
302      display.display();
303      delay(45);
304      VL53L0X_mm_previous = VL53L0X_mm_int;
305    }
306    else // en caso de superar los 300mm
307    {
308      DibujarCabecera("2_Distancia");
309      FueraDeRango();
310    }
311  }
```

La función `MedirDistanciasCortas()` mide la distancia utilizando el sensor VL53L0X, convierte esa distancia en milímetros, centímetros y pulgadas, y luego muestra el resultado en una pantalla si está dentro de un rango específico (menos de 300 mm). Si la distancia está fuera de los límites especificados, llama a una función para manejar la condición de "fuera de rango".

```
484  ISR(PCINT0_vect){
485
486    if( (PINB & B00010000) && !data_state )
487    {
488      counter ++;
489      data_state = true;
490      delay(2);
491    }
492    if( !(PINB & B00010000) && data_state )
493    {
494      counter ++;
495      data_state = false;
496      delay(2);
497    }
498
499    if(IR_enable)
500    {
501      current_count = micros();
502
503      if(PINB & B00000100)
504      {
505        if(last_IR_state == 0){
506          last_IR_state = 1;
507          width = current_count - counter_1;
508          counter_1 = current_count;
509        }
510      }
511      else if(last_IR_state == 1)
512      {
513        last_IR_state = 0;
514      }
515    }
516
517  }//fin de ISR
```

La función `ISR(PCINT0_vect)` es un manejador de interrupción, este manejador se activa cuando ocurre una interrupción en el vector `PCINT0`. Monitorea el estado del pin del puerto B bit 4(el pin 11) y actualiza la variable counter que representa el número de pasos de la rueda que utilizamos para medir la longitud y un estado (`data_state`) basado en los cambios de ese pin. Si `IR_enable` está activado, también monitorea el estado del bit 2 del puerto B (el pin 10) para medir el ancho de pulso de una señal de infrarrojos, utilizando el tiempo medido en microsegundos para calcular la duración del pulso.

```
313  void MedirLongitud(){
314    /*
315    calculamos la longitud recorrida por la rueda en cm para eso necesitamos multiplicar el
316    perimetro de la rueda dividida entre la cantidad de pasos que equivale completar una
317    vuelta por la cantidad de pasos que recorio la rueda
318    */
319    float rotary_cm = (counter * (2.5* 3.1415 / 30));
320    float rotary_mm = rotary_cm * 10;
321    float rotary_inch = rotary_cm * 0.393701;
322    int   rotary_mm_int = rotary_mm;
323
324    DibujarCabecera("3_Longitud");
325    display.setTextSize(2);
326    display.setCursor(0,10);
327    display.print("mm: ");
328    display.print(rotary_mm,0);
329    display.setCursor(0,30);
330    display.print("cm: ");
331    display.print(rotary_cm,1);
332    display.setCursor(0,50);
333    display.print("in: ");
334    display.print(rotary_inch,1);
335    display.display();
336  }
```

La función `MedirLongitud` tiene como objetivo calcular la distancia recorrida por una rueda y mostrarla en distintas unidades de medida (milímetros, centímetros y pulgadas) en una pantalla.
El perímetro de la rueda se calcula como `2.5 * 3.1415` (dónde 2.5 cm es el diámetro de la rueda y 3.1415 es una aproximación de pi), se divide por 30, que es el número de pasos necesarios para una vuelta completa de la rueda y finalmente multiplicando el número de pasos (`counter`) por este valor, se obtiene la distancia recorrida en centímetros. Posteriormente lo multiplicamos por 0.393701 para obtener el valor en pulgadas y lo dividimos entre 10 para obtener el valor en mm.

```
473  void MedirRPM(){
474    IR_enable = true;
475    int rpm = 60000000 / width;   // Los valores están en microsegundos
476    DibujarCabecera("6_RPM");
477    display.setTextSize(2);
478    display.setCursor(0,20);
479    display.print("   -RPM-");
480    display.setCursor(0,37);
481    display.print(rpm);
482    display.display();
483  }
```

La función `MedirRPM` está diseñada para calcular y mostrar las RPM (revoluciones por minuto) basándose en el ancho de pulso (`width`) medido previamente. Primero activamos la medición por infrarrojo estableciendo `IR_enable` a `true`, Posteriormente calculamos las RPM utilizando la fórmula `60000000 / width`.
`width` es el tiempo en microsegundos para una revolución completa y `60000000` es el número de microsegundos en un minuto

(60 segundos * 1,000,000 microsegundos/segundo), al dividir `60000000` por el ancho de pulso (`width`), se obtiene el número de revoluciones por minuto (RPM).

2.2 Código:

```
 ////////////////////////Librerias
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SH1106.h>
#include <VL53L0X.h>
//////////////////////

////////////////////////Pantalla OLED 64x124 con i2c
#define OLED_RESET 11
Adafruit_SH1106 display(OLED_RESET);
//////////////////////


//INPUTS/OUTPUTS
#define zero_button 6
#define mode_button 7
#define laser_button 5
#define rot_data 12
//#define rot_clock 12
#define battery_in A1
#define sharp_in A0
#define IR_rpm_in 10
#define laser_out 9


//Variables
bool mode_button_state = false;
bool zero_button_state = false;
bool laser_button_state = false;
int mode = 0;
float battery_level = 3.7;

//Para el sensor Sharp
int Sharp_cm_previous;

//Para el sensor VL53L0X
VL53L0X sensor;
int VL53L0X_mm_previous;
```

```
//Rotary encoder
int counter = 0;
bool data_state;
bool laser = false;

// MPU6050 giroscopio/acelerómetro
// Variables del giroscopio
float elapsedTime, time, timePrev;        // Variables para el
control del tiempo
int gyro_error = 0;                       // Usamos esta variable
para calcular una sola vez el error de los datos del giroscopio
float Gyr_rawX, Gyr_rawY, Gyr_rawZ;       // Aquí almacenamos los
datos sin procesar leídos del giroscopio
float Gyro_angle_x, Gyro_angle_y, Gyro_angle_z; // Aquí almacenamos
el valor del ángulo obtenido con los datos del giroscopio
// Variables del acelerómetro
int acc_error = 0;                        // Usamos esta variable
para calcular una sola vez el error de los datos del acelerómetro
float rad_to_deg = 180/3.141592654;       // Este valor es para
convertir de radianes a grados
float Acc_rawX, Acc_rawY, Acc_rawZ;       // Aquí almacenamos los
datos sin procesar leídos del acelerómetro
float Acc_angle_x, Acc_angle_y;           // Aquí almacenamos el
valor del ángulo obtenido con los datos del acelerómetro
// Ángulos totales calculados combinando datos del giroscopio y
acelerómetro
float Total_angle_x, Total_angle_y, Total_angle_z;
// Variables adicionales para corrección y control de errores
int Z_loop = 0;
bool Z_error = false;
float Gyr_rawZ_error = 0;
float angle_x_error = -6.6;
float angle_y_error = 3.6;
int print_time = 0;


// Sensor IR
// Creamos variables para los valores de tiempo de anchura de cada
señal de entrada PWM
unsigned long counter_1, current_count;
// Creamos 4 variables para almacenar el valor anterior de la señal
de entrada (si es BAJO o ALTO)
byte last_IR_state;
```

```
// Para almacenar el valor de 1000us a 2000us, creamos variables y almacenamos cada canal
int width;      // En mi caso, canal 4 del receptor y pin D12 del Arduino
bool IR_enable = false;


void setup() {

  analogReference(INTERNAL);

  //Definir los pines como salidas o entradas y establecerlos como LOW o HIGH
  pinMode(zero_button, INPUT);
  pinMode(mode_button, INPUT);
  pinMode(laser_button, INPUT);
  pinMode(rot_data, INPUT_PULLUP);
  //pinMode(rot_clock, INPUT_PULLUP);
  pinMode(battery_in, INPUT);
  pinMode(sharp_in, INPUT);
  pinMode(IR_rpm_in, INPUT);
  pinMode(laser_out, OUTPUT);
  digitalWrite(laser_out,LOW);

    PCICR |= (1 << PCIE0);    //habilitamos las interrupciones por cambio de estado en el grupo de pines 8 a 15.
    PCMSK0 |= (1 << PCINT4);  //Habilitamos la interrupción por cambio de pin específicamente para el pin 12.
    //PCMSK0 |= (1 << PCINT3);  //Habilitamos la interrupción por cambio de pin específicamente para el pin 11.
    PCMSK0 |= (1 << PCINT2);  //Habilitamos la interrupción por cambio de pin específicamente para el pin 10.

  display.begin(SH1106_SWITCHCAPVCC, 0x3C);
  // inicializamos la pantalla OLED y establecemos la comunicacion I2C entre el arduino y la pantalla.
  delay(100);
  display.clearDisplay();
  display.setTextSize(2);
  display.setCursor(0, 30);
  display.setTextColor(WHITE);
  display.println("ELECTROPRO");
  display.display();
```

```
  delay(1000);
  display.clearDisplay();
  display.display();


  //VL53L0X sensor
  sensor.init();
  sensor.setTimeout(500);
    // esta funcion establece cuanto tiempo dedica el sensor a
realizar cada medicion.
    // El tiempo es en microsegundos o sea son 200ms.
  sensor.setMeasurementTimingBudget(200000);

  //MPU6050 Giroscopio/Acelerometro
  Wire.begin();                           //configuramos el arduino
para actuar como maestro en el bus I2C
  Wire.beginTransmission(0x68);           // Iniciamos la
transmisión I2C con la dirección esclava 0x68
  Wire.write(0x6B);                       //Seleccionamos el
registro 0x6B (PWR_MGMT_1) del MPU6050
  Wire.write(0x00);                       //Escribimos 0x00 en el
registro 0x6B para desactivar el modo de reposo y activar el sensor
  Wire.endTransmission(true);             //Finaliza la transmision
I2C con el MPU6050
  //Giroscopio
  Wire.beginTransmission(0x68);           // Inicia la transmisión
I2C con la dirección esclava 0x68
  Wire.write(0x1B);                       // Selecciona el registro
0x1B (GYRO_CONFIG) del MPU6050
  Wire.write(0x10);                       // Escribe 0x10 en el
registro 0x1B para configurar el rango del giroscopio a ±1000°/modo
  Wire.endTransmission(true);             //Finaliza la transmision
I2C con el giroscopio
  //Acelerometro
  Wire.beginTransmission(0x68);           // Inicia la transmisión
I2C con la dirección esclava 0x68
  Wire.write(0x1C);                       // Selecciona el registro
0x1C (ACCEL_CONFIG) del MPU6050
  Wire.write(0x10);                       // Escribe 0x10 en el
registro 0x1C para configurar el rango del acelerómetro a ±8g
  Wire.endTransmission(true);             //  Finaliza la
transmisión I2C
  time = millis();
```

```
} //fin de la funcion setup

void loop() {
  analogReference(INTERNAL);
  battery_level = map(analogRead(battery_in),0,1024, 0, 4730);
  battery_level = battery_level / 1000;


  ////////////////////////////////CAMBIAR DE
MODO////////////////////////////////
  if(digitalRead(mode_button) && !mode_button_state)
  {
    data_state = digitalRead(rot_data);
    mode_button_state = true;
    Z_error = false;
    Z_loop = 0;
    IR_enable = false;
    Gyr_rawZ_error = 0;
    print_time = 0;  //Reinicia el tiempo de impresión para el modo
ángulo(para el MODO 3 y el MODO 4)
    mode = mode + 1;
    if(mode > 5)
    {
      mode = 0;
    }
  }
  if(!digitalRead(mode_button) && mode_button_state)
  {
    mode_button_state = false;
  }

//////////////////////////////////////////////////////////////////
//////////////


  ////////////////////////////////RESETEAR
VALOR////////////////////////////////
  if(digitalRead(zero_button) && !zero_button_state)
  {
    counter = 0;
    Z_error = false;
```

```
    Total_angle_z = 0;
    zero_button_state = true;
  }
  if(!digitalRead(zero_button) && zero_button_state)
  {
    zero_button_state = false;
  }

////////////////////////////////////////////////////////////////////
//////////////

  ////////////////////////////////LASER
BUTTON/////////////////////////////////////
  if(digitalRead(laser_button) && !laser_button_state)
  {
    laser = !laser;
    digitalWrite(laser_out, laser);
    laser_button_state = true;

  }
  if(!digitalRead(laser_button) && laser_button_state)
  {
    laser_button_state = false;
  }

////////////////////////////////////////////////////////////////////
//////////////

  switch(mode){
    case 0:  // Modo 0
      MedirDistanciasLargas();
      break;
    case 1:  // Modo 1
      MedirDistanciasCortas();
      break;
    case 2:  // Modo 2
      MedirLongitud();
      break;
    case 3:  // Modo 3
      MedirNivel();
      break;
    case 4:  // Modo 4
```

```
      MedirAngulos();
      break;
    case 5:  // Modo 5
      MedirRPM();
      break;
  }
}//fin de la funcion loop


void DibujarCabecera(String modo){
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(BLACK,WHITE);
  display.setCursor(0,0);
  display.print(modo);
  display.setTextColor(WHITE);
  display.setCursor(95,0);
  display.print(battery_level);
  display.print("V");
}

void FueraDeRango(){
  display.setTextSize(2);
  display.setCursor(0,10);
  display.print("FUERA");
  display.setCursor(0,30);
  display.print("DE");
  display.setCursor(0,50);
  display.print("RANGO");
  display.display();
}

void MedirDistanciasLargas(){
  //analogReference(DEFAULT);
  float Sharp_read = analogRead(sharp_in); // lectura del sensor
Sharp
  float Sharp_cm = pow(4727.4 / Sharp_read, 1.2134); // convertir
en cm
  int Sharp_cm_int = Sharp_cm;
  float Sharp_inch = Sharp_cm * 0.393701;
  if (Sharp_cm > 10 && Sharp_cm < 80)
  {
    DibujarCabecera("1_Distancia");
```

```
    display.setTextSize(2);
    display.setCursor(0,30);
    display.print("cm: ");
    display.print(Sharp_cm,0);
    display.setCursor(0,50);
    display.print("in: ");
    display.print(Sharp_inch,0);
    display.display();
    delay(45);
    Sharp_cm_previous = Sharp_cm_int;
  }
  if (Sharp_cm < 10)
  {
      DibujarCabecera("1_Distancia");
    display.setTextSize(2);
    display.setCursor(0,10);
    display.print("DEMASIADO");
    display.setCursor(0,30);
    display.print("CERCA");
    display.display();
  }
  if (Sharp_cm > 80)
  {
    DibujarCabecera("1_Distancia");
    FueraDeRango();
  }
}

void MedirDistanciasCortas(){
  float VL53L0X_mm = sensor.readRangeSingleMillimeters() - 40;
//40mm es un error
  float VL53L0X_cm = VL53L0X_mm / 10;
  float VL53L0X_inch = VL53L0X_cm * 0.393701;
  int VL53L0X_mm_int = VL53L0X_mm;

  if (VL53L0X_mm < 300) // siempre que la medida no supere los
300mm
  {
    DibujarCabecera("2_Distancia");
    display.setTextSize(2);
    display.setCursor(0,10);
    display.print("mm: ");
    display.print(VL53L0X_mm,0);
```

```
    display.setCursor(0,30);
    display.print("cm: ");
    display.print(VL53L0X_cm,1);
    display.setCursor(0,50);
    display.print("in: ");
    display.print(VL53L0X_inch,1);
    display.display();
    delay(45);
    VL53L0X_mm_previous = VL53L0X_mm_int;
  }
  else // en caso de superar los 300mm
  {
    DibujarCabecera("2_Distancia");
    FueraDeRango();
  }
}

void MedirLongitud(){
  /*
  calculamos la longitud recorrida por la rueda en cm para eso
necesitamos multiplicar el
  perimetro de la rueda dividida entre la cantidad de pasos que
equivale completar una
  vuelta por la cantidad de pasos que recorio la rueda
  */
  float rotary_cm = (counter * (2.5* 3.1415 / 30));
  float rotary_mm = rotary_cm * 10;
  float rotary_inch = rotary_cm * 0.393701;
  int   rotary_mm_int = rotary_mm;

  DibujarCabecera("3_Longitud");
  display.setTextSize(2);
  display.setCursor(0,10);
  display.print("mm: ");
  display.print(rotary_mm,0);
  display.setCursor(0,30);
  display.print("cm: ");
  display.print(rotary_cm,1);
  display.setCursor(0,50);
  display.print("in: ");
  display.print(rotary_inch,1);
  display.display();
}
```

```
void MedirAngulos(){
  print_time = print_time + 1;
  timePrev = time;                          // el tiempo anterior se
almacena antes de leer el tiempo actual
  time = millis();                          // se lee el tiempo
actual
  elapsedTime = (time - timePrev) / 1000; // se divide por 1000
para obtener segundos

  ////////////////////////////////////Lectura del
giroscopio//////////////////////////////////
  if (!Z_error)
  {
    DibujarCabecera("5_Angulo");
    display.setTextSize(2);
    display.setCursor(0, 30);
    display.print("CARGANDO...");
    display.display();
    while (Z_loop < 20)
    {
      timePrev = time;                          // el tiempo anterior
se almacena antes de leer el tiempo actual
      time = millis();                          // se lee el tiempo
actual
      elapsedTime = (time - timePrev) / 1000; // se divide por 1000
para obtener segundos
      Wire.beginTransmission(0x68);             // iniciar, enviar la
dirección del esclavo (en este caso 68)
      Wire.write(0x47);                         // Primera dirección
de los datos del giroscopio Z
      Wire.endTransmission(false);
      Wire.requestFrom(0x68, 2, true);          // Solicitamos solo 2
registros
      Gyr_rawZ = Wire.read() << 8 | Wire.read();
      Gyr_rawZ = (Gyr_rawZ / 32.8);
      Gyro_angle_z = Gyr_rawZ * elapsedTime;
      Gyr_rawZ_error = Gyr_rawZ_error + Gyro_angle_z;
      Z_loop = Z_loop + 1;
      delay(50);
    }
    Gyr_rawZ_error = Gyr_rawZ_error / 20;
```

```
    Z_error = true;
  }
  Wire.beginTransmission(0x68);             // iniciar, enviar la
dirección del esclavo (en este caso 68)
  Wire.write(0x47);                         // Primera dirección de
los datos del giroscopio Z
  Wire.endTransmission(false);
  Wire.requestFrom(0x68, 2, true);          // Solicitamos solo 2
registros
  Gyr_rawZ = Wire.read() << 8 | Wire.read();

  /* Ahora, para obtener los datos del giroscopio en
grados/segundos, primero tenemos que dividir
  el valor bruto por 32.8 porque ese es el valor que nos da la hoja
de datos para un rango de 1000dps */
  /*---Z---*/
  Gyr_rawZ = (Gyr_rawZ / 32.8);
  /* Ahora integramos el valor bruto en grados por segundos para
obtener el ángulo
  * Si multiplicas grados/segundos por segundos, obtienes grados */
  Gyro_angle_z = Gyr_rawZ * elapsedTime;
  Gyro_angle_z = Gyro_angle_z - Gyr_rawZ_error;
  Total_angle_z = Total_angle_z + Gyro_angle_z;

  if (print_time > 150)
  {
    DibujarCabecera("5_Angulo");
    display.setTextSize(2);
    display.setCursor(0, 30);
    display.print("X: ");
    display.print(Total_angle_z, 2); // Imprimo valores de Y para X
debido a cómo está colocado el IMU dentro de la carcasa
    display.display();
  }
}

void MedirNivel(){
  print_time = print_time + 1;
  timePrev = time;                          // el tiempo anterior se
almacena antes de leer el tiempo actual
  time = millis();                          // se lee el tiempo
actual
```

```
  elapsedTime = (time - timePrev) / 1000; // se divide por 1000
para obtener segundos

  ////////////////////////////////////Lectura del
giroscopio//////////////////////////////////////
  Wire.beginTransmission(0x68);             // iniciar, enviar la
dirección del esclavo (en este caso 68)
  Wire.write(0x43);                         // Primera dirección de
los datos del giroscopio
  Wire.endTransmission(false);
  Wire.requestFrom(0x68, 4, true);          // Solicitamos solo 4
registros

  Gyr_rawX = Wire.read() << 8 | Wire.read(); // Nuevamente
desplazamos y sumamos
  Gyr_rawY = Wire.read() << 8 | Wire.read();
  /* Ahora, para obtener los datos del giroscopio en
grados/segundos, primero tenemos que dividir
  el valor bruto por 32.8 porque ese es el valor que nos da la hoja
de datos para un rango de 1000dps */
  /*---X---*/
  Gyr_rawX = (Gyr_rawX / 32.8);
  /*---Y---*/
  Gyr_rawY = (Gyr_rawY / 32.8);

  /* Ahora integramos el valor bruto en grados por segundos para
obtener el ángulo
  * Si multiplicas grados/segundos por segundos, obtienes grados */
  /*---X---*/
  Gyro_angle_x = Gyr_rawX * elapsedTime;
  /*---Y---*/
  Gyro_angle_y = Gyr_rawY * elapsedTime;

  ////////////////////////////////////Lectura del
acelerómetro//////////////////////////////////////
  Wire.beginTransmission(0x68);     // iniciar, enviar la dirección
del esclavo (en este caso 68)
  Wire.write(0x3B);                 // Solicitar el registro 0x3B -
corresponde a AcX
  Wire.endTransmission(false);      // mantener la transmisión y
continuar
  Wire.requestFrom(0x68, 6, true);  // Solicitamos los siguientes 6
registros comenzando con el 3B
```

```
  /* Hemos solicitado el registro 0x3B. El IMU enviará una ráfaga
de registros.
  La cantidad de registros a leer se especifica en la función
requestFrom.
  En este caso, solicitamos 6 registros. Cada valor de aceleración
se compone de
  dos registros de 8 bits, valores bajos y valores altos. Por eso
solicitamos los 6
  y sumamos cada par. Para eso desplazamos a la izquierda el
registro de valores altos (<<)
  y hacemos una operación OR (|) para añadir los valores bajos.
  Si leemos la hoja de datos, para un rango de ±8g, tenemos que
dividir los valores brutos por 4096 */
  Acc_rawX = (Wire.read() << 8 | Wire.read()) / 4096.0; // cada
valor necesita dos registros
  Acc_rawY = (Wire.read() << 8 | Wire.read()) / 4096.0;
  Acc_rawZ = (Wire.read() << 8 | Wire.read()) / 4096.0;
  /* Ahora, para obtener los ángulos del acelerómetro, usamos la
fórmula de Euler con los valores de aceleración
  y luego restamos el valor de error encontrado anteriormente */
  /*---X---*/
  Acc_angle_x = (atan((Acc_rawY) / sqrt(pow((Acc_rawX), 2) +
pow((Acc_rawZ), 2))) * rad_to_deg);
  /*---Y---*/
  Acc_angle_y = (atan(-1 * (Acc_rawX) / sqrt(pow((Acc_rawY), 2) +
pow((Acc_rawZ), 2))) * rad_to_deg);
  ///////////////////////////////////Ángulo total y
filtro/////////////////////////////////
  /*---Ángulo del eje X---*/
  Total_angle_x = 0.98 * (Total_angle_x + Gyro_angle_x) + 0.02 *
Acc_angle_x;
  /*---Ángulo del eje Y---*/
  Total_angle_y = 0.98 * (Total_angle_y + Gyro_angle_y) + 0.02 *
Acc_angle_y;

  float x_without_error = (Total_angle_x - angle_x_error) * -1;
  float y_without_error = (Total_angle_y - angle_y_error);

  if (print_time > 150)
  {
    DibujarCabecera("4_Nivel");
    display.setTextSize(2);
    display.setCursor(0, 25);
```

```
    display.print("X: ");
    display.print(y_without_error, 1); // Imprimo valores de Y para
X debido a cómo está colocado el IMU dentro de la carcasa
    display.setCursor(0, 45);
    display.print("Y: ");
    display.print(x_without_error, 1); // Imprimo valores de X para
Y debido a cómo está colocado el IMU dentro de la carcasa
    display.display();
  }
}


void MedirRPM(){
  IR_enable = true;
  int rpm = 60000000 / width;    // Los valores están en
microsegundos
  DibujarCabecera("6_RPM");
  display.setTextSize(2);
  display.setCursor(0,20);
  display.print("   -RPM-");
  display.setCursor(0,37);
  display.print(rpm);
  display.display();
}

ISR(PCINT0_vect){

  if( (PINB & B00010000) && !data_state )
  {
    counter ++;
    data_state = true;
    delay(2);
  }
  if( !(PINB & B00010000) && data_state )
  {
    counter ++;
    data_state = false;
    delay(2);
  }

  if(IR_enable)
  {
    current_count = micros();
```

```
    if(PINB & B00000100)
    {
      if(last_IR_state == 0){
        last_IR_state = 1;
        width = current_count - counter_1;
        counter_1 = current_count;
      }
    }
    else if(last_IR_state == 1)
    {
      last_IR_state = 0;
    }
  }

}//fin de ISR
```

3. ANEXO

3.1. Información técnica

3.1.1 ARDUINO NANO

El Arduino Nano es una placa compacta de microcontrolador programable basada en el ATmega328 (Arduino Nano 3.x) o ATmega168 (Arduino Nano original). Es una versión más pequeña y ligera del Arduino Uno, diseñada para ser usada en prototipos y proyectos donde el espacio es limitado. Aquí tienes información técnica detallada sobre el Arduino Nano:

Especificaciones Técnicas

Microcontrolador

- **Modelo**: ATmega328P (Nano 3.x) o ATmega168 (Nano original)
- **Arquitectura**: AVR de 8 bits

Voltajes de Funcionamiento

- **Voltaje de Operación**: 5V
- **Voltaje de Entrada (recomendado)**: 7-12V
- **Voltaje de Entrada (límites)**: 6-20V

Pines de Entrada/Salida

- **Pines Digitales I/O**: 14 (de los cuales 6 pueden ser utilizados como salidas PWM)
- **Pines Analógicos**: 8
- **Corriente DC por pin I/O**: 40 mA
- **Corriente DC para pin 3.3V**: 50 mA

Memoria

- **Memoria Flash**: 32 KB (ATmega328) de los cuales 2 KB son usados por el bootloader
- **SRAM**: 2 KB (ATmega328)
- **EEPROM**: 1 KB (ATmega328)

Velocidad de Reloj

- **Frecuencia**: 16 MHz

Conectividad

- **Interfaz USB**: Mini-B USB
- **Conector ISP**: Disponible para programación directa

Características Físicas

- **Dimensiones**: 18 x 45 mm
- **Peso**: 7 gramos aproximadamente

Pines y Conectores

- **Pines de Alimentación**:
  - **VCC**: Voltaje regulado de 5V
  - **GND**: Tierra
  - **VIN**: Voltaje de entrada al regulador, que puede ser de 7 a 12V
  - **3V3**: Salida de 3.3V generada por un regulador a bordo
- **Pines de I/O Digitales**:
  - **D0-D13**: Pines digitales con funciones de entrada y salida
  - **PWM**: Pines D3, D5, D6, D9, D10, D11 pueden funcionar como salidas PWM
- **Pines Analógicos**:
  - **A0-A7**: Entradas analógicas
- **Otros Pines**:
  - **AREF**: Referencia de voltaje para entradas analógicas
  - **Reset**: Para reiniciar el microcontrolador

Programación

El Arduino Nano puede ser programado utilizando el entorno de desarrollo Arduino (IDE) que es compatible con Windows, macOS y Linux. La programación se realiza a través del conector USB utilizando el bootloader incluido en el microcontrolador. También puede ser programado utilizando un programador externo a través del conector ISP.

Comunicaciones

- **Serial**: UART (pines D0 (RX) y D1 (TX))
- **I2C**: Soporte a través de los pines A4 (SDA) y A5 (SCL)
- **SPI**: Soporte a través de los pines D10 (SS), D11 (MOSI), D12 (MISO), y D13 (SCK)

Para más información, datasheet:
https://docs.arduino.cc/resources/datasheets/A000005-datasheet.pdf

3.1.2 Pantalla OLED SH1106 128x64

La pantalla OLED SH1106 es un módulo de visualización que utiliza un controlador SH1106 para gestionar una matriz de 128x64 píxeles. Es popular en aplicaciones de electrónica debido a su alta calidad de imagen, bajo consumo de energía y facilidad de uso. Aquí tienes la información técnica detallada de la pantalla OLED SH1106 de 1.3 pulgadas y 128x64 píxeles:

Especificaciones Técnicas

Dimensiones

- **Tamaño de la Pantalla**: 1.3 pulgadas
- **Resolución**: 128x64 píxeles

Características del OLED

- **Tipo de Display**: OLED (Diodo Orgánico Emisor de Luz)
- **Color**: Monocromático (normalmente blanco, azul o amarillo)
- **Contraste**: Alto contraste y amplia gama de ángulos de visión
- **Retroiluminación**: No requiere, ya que los píxeles emiten luz propia

Controlador

- **Controlador**: SH1106
- **Interfaz de Comunicación**: I2C, SPI (dependiendo del módulo específico)

Conectividad

- **Interfaz I2C**:
  - **Pines**: SCL (reloj) y SDA (datos)
  - **Dirección I2C**: Generalmente 0x3C o 0x3D, dependiendo del módulo
- **Interfaz SPI**:
  - **Pines**: CS (Chip Select), DC (Data/Command), RES (Reset), D0 (Clock), D1 (Data)

Voltajes de Operación

- **Voltaje de Alimentación**: 3.3V a 5V
- **Corriente de Operación**: Aproximadamente 20-30 mA

Pines y Conexiones

Para la versión con interfaz I2C:

- **GND**: Tierra
- **VCC**: Voltaje de alimentación (3.3V o 5V)
- **SCL**: Reloj I2C
- **SDA**: Datos I2C

Para la versión con interfaz SPI:

- **GND**: Tierra
- **VCC**: Voltaje de alimentación (3.3V o 5V)
- **CS**: Chip Select
- **DC**: Data/Command
- **RES**: Reset
- **D0**: Clock
- **D1**: Data

Características Adicionales

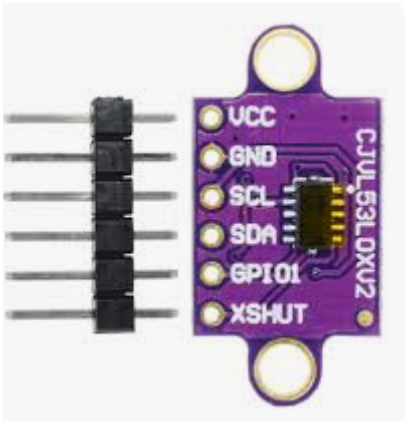
- **Ángulo de Visión**: Amplio (cerca de 160 grados)
- **Temperatura de Operación**: Generalmente de -40°C a 70°C
- **Vida Útil**: Alta durabilidad con más de 50,000 horas de operación

Programación y Uso

La pantalla OLED SH1106 se puede programar y controlar utilizando diversas bibliotecas disponibles para diferentes plataformas de desarrollo, como Arduino, Raspberry Pi y otros microcontroladores.

Para más información, datasheet: https://www.pololu.com/file/0J1813/SH1106.pdf

3.1.3 VL53L0X



El VL53L0X es un sensor de distancia basado en la tecnología Time-of-Flight (ToF) desarrollado por STMicroelectronics. Este sensor es conocido por su precisión y capacidad para medir distancias de forma rápida y eficiente. A continuación, se proporciona información técnica detallada sobre el VL53L0X:

Especificaciones Técnicas

General

- **Tipo de Sensor**: Rango de distancia óptico basado en Time-of-Flight (ToF)
- **Tecnología**: LIDAR (Light Detection and Ranging)
- **Modelo**: VL53L0X

Capacidades de Medición

- **Rango de Medición**:
  - **Rango Corto**: 30 mm a 2 m (depende de las condiciones de iluminación y del objeto)
  - **Rango Largo**: Hasta 2 metros en condiciones óptimas
- **Precisión**: ±3% en el rango típico
- **Resolución**: 1 mm

Características del Sensor

- **Ángulo de Visión**: Aproximadamente 25 grados
- **Tasa de Actualización**: Hasta 50 Hz (ajustable según la precisión y el rango deseado)
- **Temperatura de Operación**: -20°C a +70°C

Conectividad

- **Interfaz de Comunicación**: I2C (Inter-Integrated Circuit)
- **Dirección I2C**: 0x29 (por defecto, puede ser modificada)

Alimentación

- **Voltaje de Operación**: 2.6V a 3.5V
- **Corriente de Operación**: 10 mA (típico en modo de medición continuo)

Pines y Conexiones

El VL53L0X generalmente se encuentra en módulos que facilitan su uso. Un módulo típico incluye los siguientes pines:

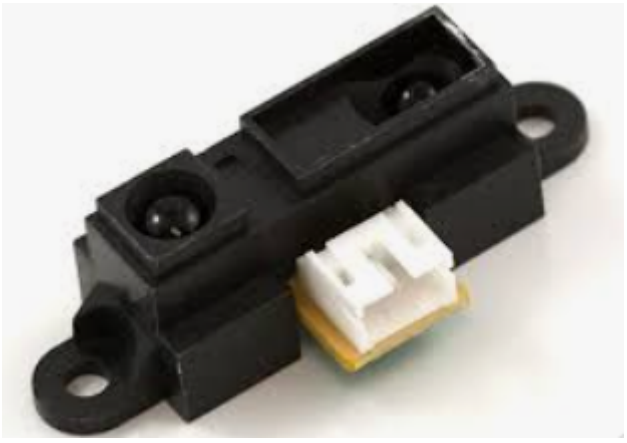
- **VIN**: Voltaje de entrada (2.6V a 5V, con regulación interna)
- **GND**: Tierra
- **SCL**: Reloj I2C
- **SDA**: Datos I2C
- **XSHUT**: Pin de apagado (opcional)
- **GPIO1**: Pin de interrupción (opcional, puede ser usado para señalizar mediciones completadas)

Características Adicionales

- **Bajo Consumo de Energía**: Optimizado para aplicaciones móviles y de batería.
- **Alta Velocidad de Medición**: Adecuado para aplicaciones en tiempo real.
- **Pequeño Tamaño**: Adecuado para aplicaciones con restricciones de espacio.

Para más información, datasheet: **https://www.st.com/resource/en/datasheet/vl53l0x.pdf**

3.1.4 SHARP 2Y0A21



El SHARP 2Y0A21 es un sensor de distancia infrarrojo que utiliza un método de triangulación para calcular la distancia entre el sensor y un objeto. Es especialmente popular en aplicaciones de robótica y automatización debido a su facilidad de uso y fiabilidad. A continuación, te proporciono información técnica detallada sobre este sensor:

Especificaciones Técnicas

General

- **Tipo de Sensor: Sensor de distancia infrarrojo**
- **Modelo: SHARP 2Y0A21**

Capacidades de Medición

- **Rango de Medición: 10 cm a 80 cm (4 pulgadas a 32 pulgadas)**
- **Precisión: Depende de las condiciones ambientales y la reflectividad del objeto, generalmente alrededor del 10%**
- **Ángulo de Visión: Aproximadamente 25 grados**

Características del Sensor

- **Método de Medición: Triangulación mediante emisión y recepción de luz infrarroja**
- **Tipo de Salida: Analógica (voltaje proporcional a la distancia medida)**
- **Tiempo de Respuesta: Rápido, generalmente en el orden de milisegundos**
- **Temperatura de Operación: -10°C a +60°C**

Alimentación

- **Voltaje de Operación: 4.5V a 5.5V**
- **Corriente de Operación: Aproximadamente 30 mA**

Pines y Conexiones

El sensor SHARP 2Y0A21 generalmente tiene tres pines:

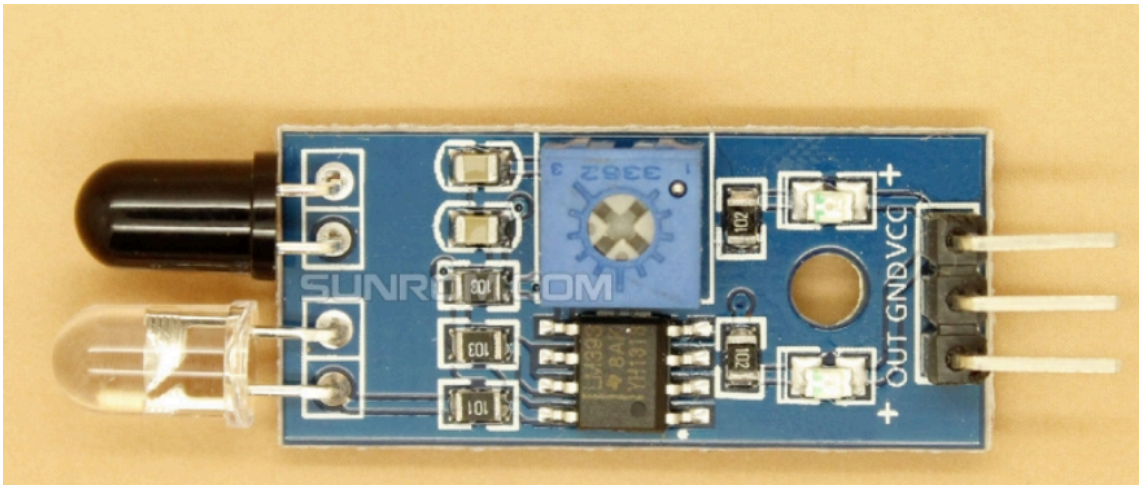
- **VCC: Alimentación (entre 4.5V y 5.5V)**
- **GND: Tierra**
- **Vo: Salida analógica (voltaje proporcional a la distancia medida)**

Características Adicionales

- **Robusto y Confiable: Diseñado para aplicaciones industriales y de robótica.**
- **Fácil de Interfaz: Solo requiere una conexión analógica y alimentación para su funcionamiento.**
- **Compacto y Ligero: Fácil de integrar en diferentes diseños y proyectos.**

Para más información, datasheet: Para más información, datasheet:
https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf

3.1.5 IR module



Características del módulo de sensor de infrarrojos

- Tensión de funcionamiento de 5 VCC
- Los pines de E/S cumplen con 5 V y 3,3 V.
- Alcance: hasta 20 cm
- Rango de detección ajustable
- Sensor de luz ambiental incorporado
- corriente de suministro de 20 mA
- Orificio de montaje

Breve información sobre el módulo de sensor de infrarrojos

El módulo del sensor de infrarrojos consta principalmente del transmisor y receptor de infrarrojos, el amplificador operacional, la resistencia variable (potenciómetro de ajuste), el LED de salida y algunas resistencias.

Transmisor LED de infrarrojos

El LED IR emite luz en el rango de frecuencia infrarroja. La luz IR es invisible para nosotros porque su longitud de onda (700 nm – 1 mm) es mucho mayor que el rango de luz visible. Los LED IR tienen un ángulo de emisión de luz de aprox. 20-60 grados y rango de aprox. Desde unos centímetros hasta varios pies, depende del tipo de transmisor de infrarrojos y del fabricante. Algunos transmisores tienen un alcance en kilómetros. LED IR de color blanco o transparente, por lo que puede emitir la máxima cantidad de luz.

Receptor de fotodiodo

El fotodiodo actúa como receptor de infrarrojos y conduce cuando la luz incide sobre él. El fotodiodo es un semiconductor que tiene una unión PN, operada en polarización inversa, lo que significa que comienza a conducir la corriente en dirección inversa cuando la luz incide sobre él, y la cantidad de flujo de corriente es proporcional a la cantidad de luz. Esta propiedad lo

hace útil para la detección de infrarrojos. El fotodiodo parece un LED, con una capa de color negro en su lado exterior. El color negro absorbe la mayor cantidad de luz.

Amplificador operacional LM358

LM358 es un amplificador operacional (Op-Amp) que se utiliza como comparador de voltaje en el sensor de infrarrojos. el comparador comparará el voltaje umbral establecido usando el valor preestablecido (pin2) y el voltaje de resistencia en serie del fotodiodo (pin3).

Caída de voltaje de la resistencia en serie del fotodiodo> Voltaje umbral = La salida del amplificador operacional es alta

Caída de voltaje de la resistencia en serie del fotodiodo <Voltaje umbral = La salida del amplificador operacional es baja

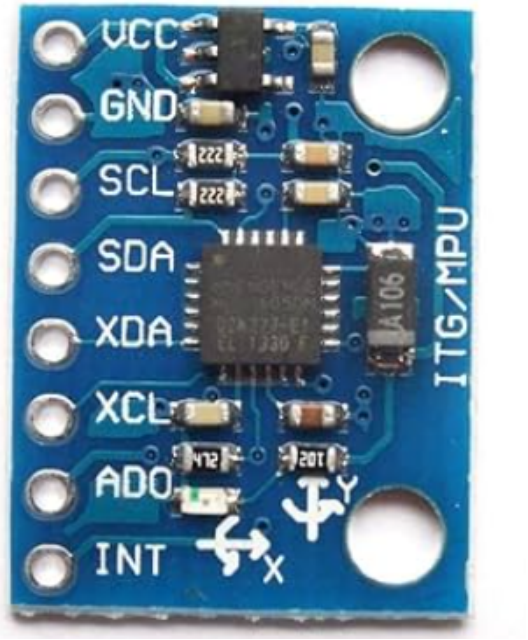
Cuando la salida de Opamp es alta, el LED en el terminal de salida de Opamp se enciende (lo que indica la detección de un objeto).

Resistencia variable

La resistencia variable utilizada aquí es una preestablecida. Se utiliza para calibrar el rango de distancia en el que se debe detectar el objeto.

Para más información, datasheet: Este componente no tiene datasheet

3.1.6 MPU6050



El MPU6050 es un módulo de sensor de movimiento de 6 grados de libertad (6-DoF) que combina un acelerómetro y un giroscopio en un solo chip. Fabricado por InvenSense (ahora parte de TDK), este módulo es ampliamente utilizado en proyectos de electrónica y robótica para detectar y medir la orientación, la aceleración y la rotación de un objeto. A continuación, te proporciono información técnica detallada sobre el MPU6050:

Especificaciones Técnicas

General

- **Tipo de Sensor**: Acelerómetro y Giroscopio de 6 ejes
- **Modelo**: MPU6050

Acelerómetro

- **Rango de Medición**: ±2g, ±4g, ±8g, ±16g
- **Sensibilidad**: Configurable dependiendo del rango seleccionado
- **Resolución**: 16 bits (valores digitales)
- **Frecuencia de Muestreo**: Hasta 1 kHz

Giroscopio

- **Rango de Medición**: ±250°/s, ±500°/s, ±1000°/s, ±2000°/s
- **Sensibilidad**: Configurable dependiendo del rango seleccionado
- **Resolución**: 16 bits (valores digitales)
- **Frecuencia de Muestreo**: Hasta 8 kHz

Características del Sensor

- **Interfaz de Comunicación**: I2C (Inter-Integrated Circuit)
- **Dirección I2C**: 0x68 (dirección predeterminada) o 0x69 (con pin AD0 conectado a Vcc)
- **Temperatura de Operación**: -40°C a +85°C

Alimentación

- **Voltaje de Operación**: 3.3V (típicamente)
- **Corriente de Operación**: 3.6 mA (Acelerómetro activo y giroscopio en reposo)
- **Corriente en Reposo**: 5 µA (Modo de bajo consumo)

Otros

- **Interrupciones**: Soporte para interrupciones programables
- **Detección de Movimiento**: Capacidad para detectar eventos de movimiento y orientación

Pines y Conexiones

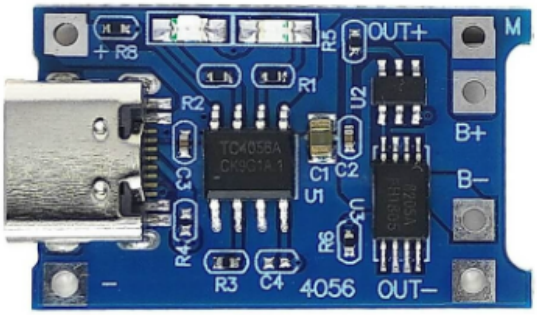
El MPU6050 generalmente tiene 8 pines, incluyendo VCC, GND, SDA, SCL, INT, AD0, y los pines de alimentación para el acelerómetro y el giroscopio.

Características Adicionales

- **FIFO**: Buffer FIFO integrado para almacenar datos de sensor
- **DMP (Digital Motion Processor)**: Procesador digital de movimiento integrado para realizar cálculos de fusión de sensores y detección de movimiento

Para más información, datasheet:
https://cdn.sparkfun.com/datasheets/Sensors/Accelerometers/RM-MPU-6000A.pdf

3.1.7 TP4056



El TP4056 es un controlador de carga de batería de litio altamente integrado y económico, utilizado en una amplia variedad de dispositivos alimentados por baterías de litio, como teléfonos móviles, cámaras, dispositivos portátiles, etc. Proporciona una solución de carga simple y confiable para baterías de litio de una sola celda. A continuación, te proporciono información técnica detallada sobre el TP4056:

Especificaciones Técnicas

General

- **Tipo de Controlador: Controlador de carga de batería de litio**
- **Modelo: TP4056**

Capacidad de Carga

- **Voltaje de Entrada: 4.5V - 5.5V (típicamente alimentado a través de un puerto USB)**
- **Corriente de Carga: 1A (ajustable con resistencias externas)**
- **Tipo de Batería: Baterías de litio de una sola celda (Li-Ion o Li-Po)**
- **Voltaje de Carga: 4.2V (típicamente)**

Protecciones

- **Protección contra Sobrecarga: El TP4056 interrumpe la carga cuando la batería alcanza el voltaje máximo de carga (4.2V)**
- **Protección contra Sobredescarga: Protege la batería de la descarga excesiva durante la operación.**
- **Protección contra Cortocircuitos: Evita daños en el circuito debido a cortocircuitos.**

Características del Controlador

- **Método de Control: Control de corriente constante / voltaje constante**
- **Eficiencia: Alto rendimiento de carga con bajo consumo de energía**
- **Temperatura de Operación: -20°C a +85°C**

Pines y Conexiones

El TP4056 generalmente tiene 8 pines:

- **B+: Terminal positivo de la batería**
- **B-: Terminal negativo de la batería**
- **PROG: Pin de programación de corriente de carga (conexión a resistencias externas para ajustar la corriente de carga)**
- **BAT: Salida de estado de carga de la batería**
- **OUT: Salida de carga**
- **GND: Tierra**
- **+: Terminal de entrada positiva**
- **-: Terminal de entrada negativa**

Características Adicionales

- **LED Indicador: Algunos módulos TP4056 vienen con un LED que indica el estado de carga (rojo para carga, verde para carga completa).**
- **Pequeño Tamaño: Fácil de integrar en diferentes diseños y proyectos.**
- **Bajo Costo: Solución de carga económica para proyectos de electrónica DIY.**

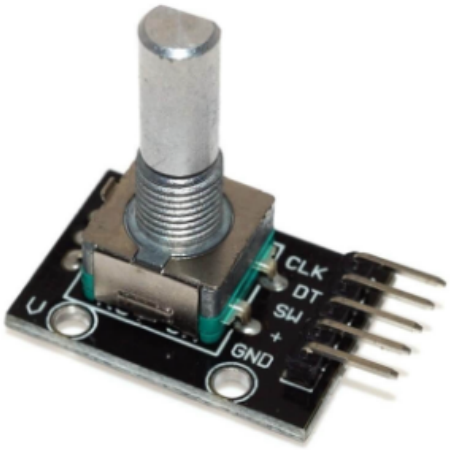
Uso y Aplicaciones

El TP4056 se utiliza en una amplia gama de aplicaciones donde se requiere carga de batería de litio de una sola celda, como:

- **Cargadores portátiles**
- **Linternas LED recargables**
- **Dispositivos de IoT alimentados por batería**
- **Proyectos de electrónica DIY**

Para más información, datasheet:
https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Prototyping/TP4056.pdf

3.1.8 KY040(Encoder rotativo)



El KY-040 es un módulo de codificador rotativo de tipo incremental, comúnmente utilizado para detectar el movimiento y la dirección de rotación de un eje. Es popular en proyectos de electrónica y robótica donde se necesita control de entrada de usuario o detección de movimiento rotativo. Aquí tienes información técnica detallada sobre el módulo KY-040:

Especificaciones Técnicas

General

- **Tipo de Sensor**: Módulo de codificador rotativo
- **Modelo**: KY-040

Funcionamiento

- **Tipo de Codificación**: Incremental
- **Mecanismo de Codificación**: Cuenta los pulsos de dos señales cuadradas en cuadratura (A y B) generadas por el giro del eje.
- **Dirección de Rotación**: La dirección de rotación se determina comparando los estados de las señales A y B.

Características del Módulo

- **Voltaje de Alimentación**: 5V DC
- **Corriente de Operación**: <20 mA
- **Velocidad Máxima de Rotación**: Generalmente, depende de la frecuencia de lectura y la capacidad de procesamiento del microcontrolador.
- **Tipo de Salida**: Digital (pulsos de A y B) o Pulsos y Dirección

Pines y Conexiones

El módulo KY-040 generalmente tiene 5 pines:

- **CLK (A)**: Señal de pulso A generada por el giro del eje.
- **DT (B)**: Señal de pulso B generada por el giro del eje.
- **SW (Switch)**: Interruptor pulsador integrado (opcional).
- **+**: Alimentación (5V)
- **-**: Tierra (GND)

Uso y Aplicaciones

El KY-040 se utiliza en una variedad de proyectos donde se requiere entrada de usuario o detección de movimiento rotativo, como:

- Control de volumen en dispositivos electrónicos
- Selección de opciones en pantallas LCD
- Control de desplazamiento en pantallas de desplazamiento
- Control de motores paso a paso y servomotores

Para más información, datasheet:
https://www.alldatasheet.es/datasheet-pdf/pdf/1648739/JOY-IT/KY-040.html

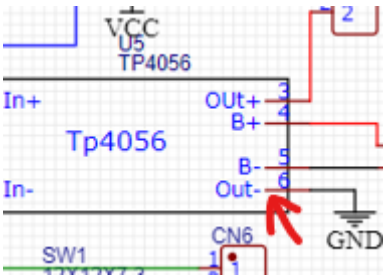
3.2. Protocolo de test

- Para realizar las mediciones se llevarán a cabo los siguientes puntos:
  - Se alimentará el circuito con tensión continua de 5 voltios a través del conector de carga.
  - Se poseerá un multímetro para realizar las mediciones.
- Descripción de los puntos de prueba(Test point) del circuito:

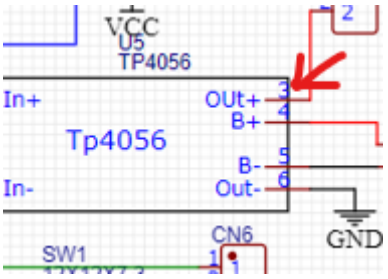
Punto de test	Descripción
TP1	GND, punto de referencia negativo
TP2	OUT+ del TP4056
TP3	PIN de la alimentación de la pantalla
TP4	PIN B+ de la batería

- **ubicación de los puntos de prueba:**

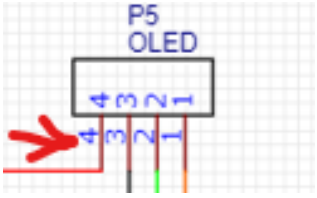
TP1 GND



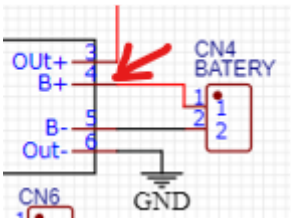
TP2 OUT+



TP3 VCC de pantalla



TP4 PIN B+ de la batería



3.3. Documentación para el Servicio de Asistencia Técnica

Cada medición se hará tomando como referencia la negativa TP1

Síntoma	Solución
El dispositivo no enciende	Comprobar la alimentación de 5V(TP2)
La pantalla no enciende	Comprobar la alimentación de la pantalla “5V” (TP3)
La batería no carga	Comprobar si hay alimentación en B+(TP4)

3.4. Documentación para el cliente

Con el dispositivo de medición múltiple de Electropro puedes realizar medidas de forma sencilla, medir distancias cortas con precisión, medir distancias de hasta 80cm, medir la longitud de una superficie pasando una rueda también puedes medir ángulos, nivel y utilizarlo como tacómetro para medir RPM, todo esto en un solo dispositivo.

- **Puesta en marcha**

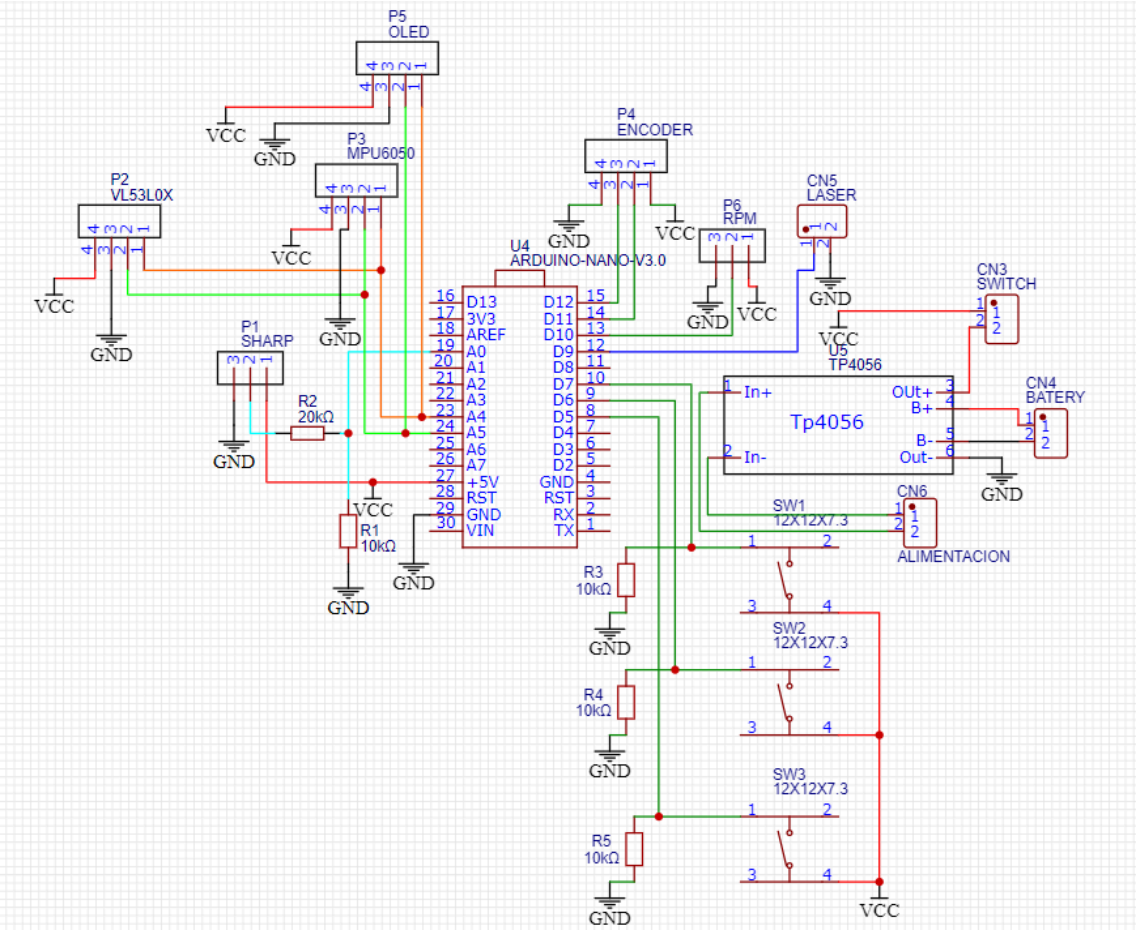
1. Primero nos aseguramos de que el dispositivo contiene su batería y este cargado.
2. Encendemos el dispositivo con el interruptor deslizante.
3. Elegimos el modo que deseamos con el pulsador amarillo (cada vez que pulsemos el pulsador cambiara al siguiente modo).
4. Procedemos a realizar la medida, en caso de querer realizar alguna nueva medida pulsamos el pulsador azul.

- **Resolucion de averias**

Síntoma	Solución
El dispositivo no enciende	-Comprobar si la batería está cargada -Comprobar la posición del switch deslizante y si esta bien conectado en la placa
La pantalla está distorsionada	-Comprobar que la pantalla este bien conectada a la placa
El láser no enciende	-Comprobar que el módulo láser esté bien conectada a la placa
uno de los sensores no funciona	-Comprobar que el sensor láser esté bien conectada a la placa

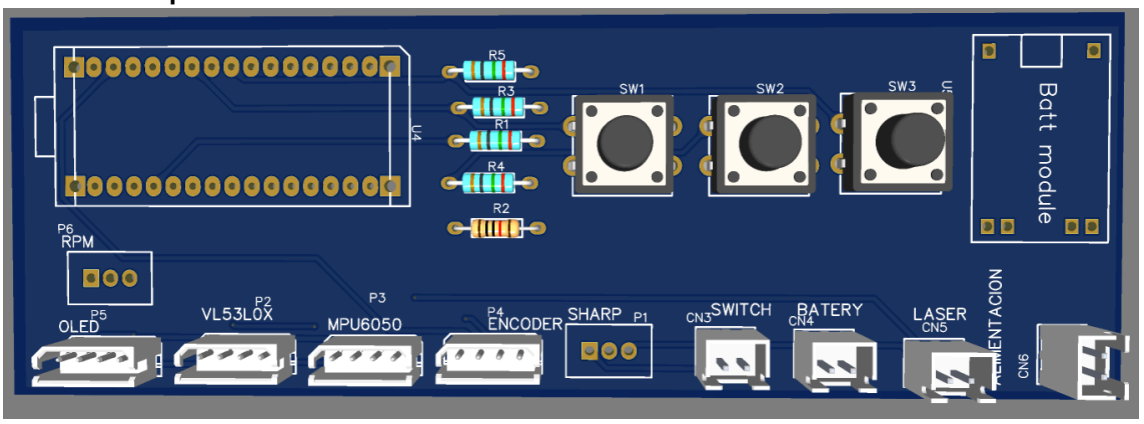
4. PLANOS Y ESQUEMAS

4.1. Esquemas electrónicos

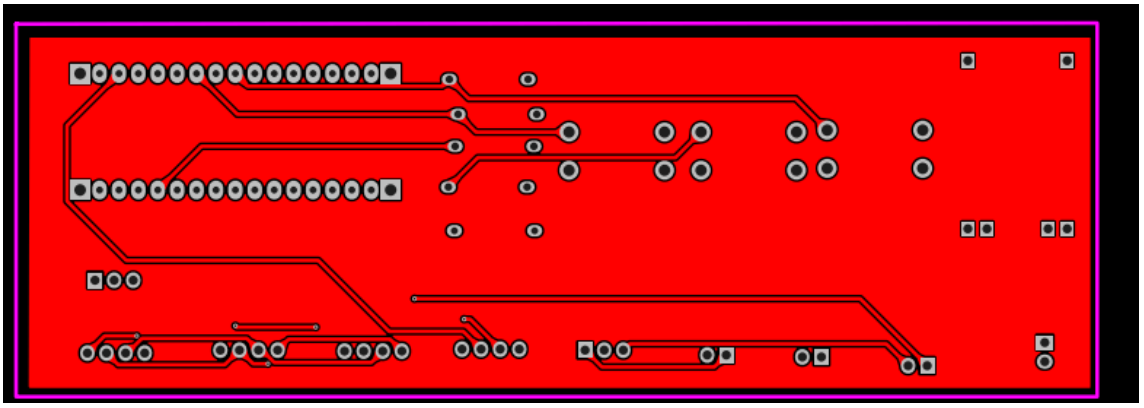


4.1. Circuitos impresos. Capas

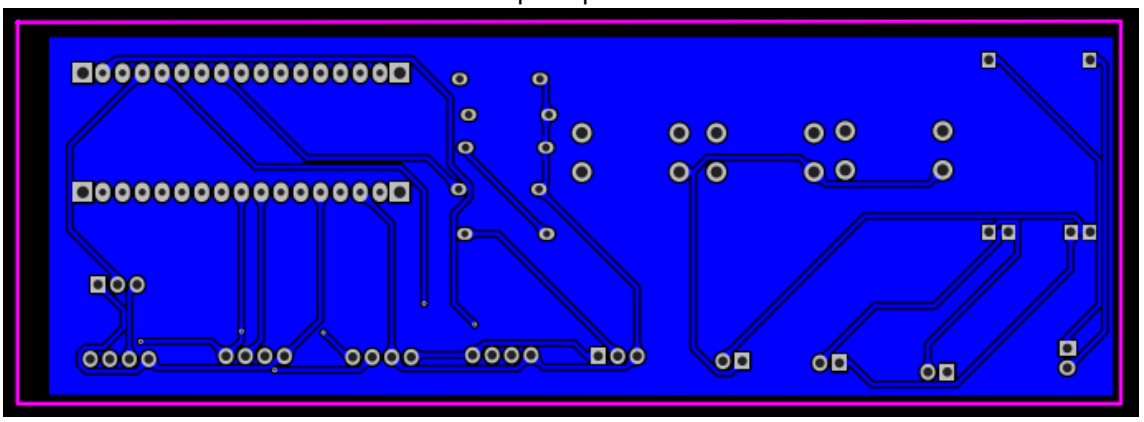
4.1.1. Componentes - TOP



4.1.2. Pistas - BOT

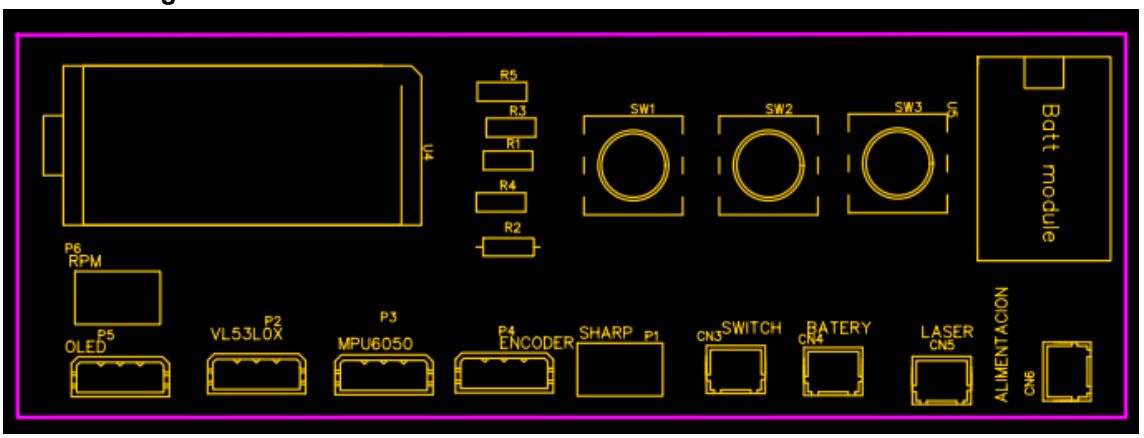


Capa superior

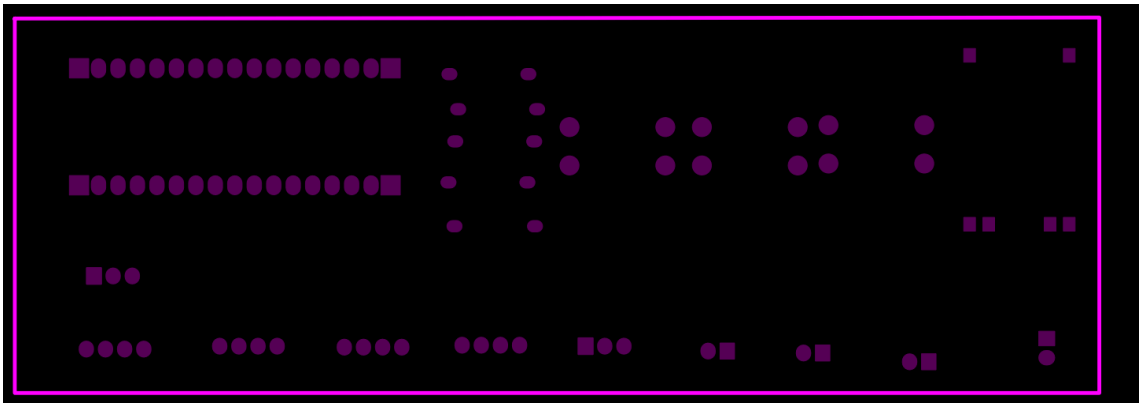


Capa inferior

4.1.3. Serigrafia - SST



4.1.3. Plano de taladros -DRD



4.2. Informes

4.2.1. Lista de componentes

Componentes	Modelo	Cantidad
Arduino	Nano	1
Pantalla OLED 128x64	SH1106	1
sensor de distancia	VL53L0X	1
sensor de distancia	Sharp 2Y0A21	1
acelerómetro/giroscopio	MPU6050	1
controlador de carga de batería	TP4056	1
codificador rotativo	KY-040	1
conectores JST-XH	4 pines 2.54mm	4
conectores JST-XH	3 pines 2.54mm	2
conectores JST-XH	2 pines 2.54mm	4
Pulsadores	12x12x7.3mm	3
Módulo IR	—	1
Resistencias	10KOhm	4
Resistencia	22K	1
Termoretráctiles	1.0x45mm	4
Termoretráctiles	1.5x45mm	2
cables	22 awg	90cm

4.2.2. Cinta de Taladrado

```
M48
METRIC,LZ,000.000
;FILE_FORMAT=3:3
;TYPE=PLATED
;Layer: PTH_Through
;EasyEDA v6.5.42, 2024-06-05 20:39:10
;c586001c169e40d7ba6ae46cb4bdde6a,0c9828c0c674422f84df0e41d11abf93,10
;Gerber Generator version 0.2
;Holesize 1 = 0.306 mm
T01C0.306
;Holesize 2 = 0.915 mm
T02C0.915
;Holesize 3 = 1.000 mm
T03C1.000
;Holesize 4 = 1.200 mm
T04C1.200
;Holesize 5 = 1.501 mm
T05C1.501
;Holesize 6 = 1.524 mm
T06C1.524
%
G05
G90
T01
X055118Y-004566
X065637Y-004742
X042207Y-005882
X078557Y-001019
X085049Y-003695
X059379Y-009534
T02
X083777Y007874
X094277Y007874
X072898Y028448
X070358Y028448
X067818Y028448
X065278Y028448
X062738Y028448
X060198Y028448
X057658Y028448
X055118Y028448
X052578Y028448
X050038Y028448
X047498Y028448
X044958Y028448
X042418Y028448
X039878Y028448
X037338Y028448
X037338Y013208
X039878Y013208
X042418Y013208
X044958Y013208
X047498Y013208
X050038Y013208
X052578Y013208
X055118Y013208
X057658Y013208
X060198Y013208
```

X062738Y013208
X065278Y013208
X067818Y013208
X070358Y013208
X072898Y013208
T03
X083877Y018923
X094177Y018923
X084258Y023114
X094558Y023114
X082988Y013589
X093288Y013589
X083115Y027686
X093415Y027686
T04
X119360Y-008382
X116860Y-008382
X131806Y-008636
X129306Y-008636
X145649Y-009779
X143149Y-009779
X161036Y-006751
X161036Y-009251
X053147Y-007747
X055646Y-007747
X058146Y-007747
X060645Y-007747
X069403Y-007874
X071902Y-007874
X074402Y-007874
X076901Y-007874
X084770Y-007620
X087269Y-007620
X089769Y-007620
X092268Y-007620
X035748Y-008128
X038247Y-008128
X040747Y-008128
X043246Y-008128
X150971Y030131
X163969Y030129
X163971Y008131
X161471Y008131
X150970Y008131
X153469Y008129
X100878Y-007747
X103378Y-007747
X105878Y-007747
X036743Y001397
X039243Y001397
X041743Y001397
T05
X111279Y015788
X098779Y015788
X111279Y020788
X098779Y020788
X128551Y015788
X116051Y015788
X128551Y020788
X116051Y020788

X145061Y016042
X132561Y016042
X145061Y021042
X132561Y021042
T06
X075438Y013208
X075438Y028448
X034798Y028448
X034798Y013208
M30

5. PLIEGO DE CONDICIONES

5.1. Normativa de obligado cumplimiento

- UNE 20-050-74 (I). Código para las marcas de resistencias y condensadores. Valores y tolerancias.

- UNE 20-524-75 (I). Técnica circuitos impresos. Parámetros fundamentales. Sistemas de cuadrícula.

- UNE 20-524. Equipos electrónicos y sus componentes. Soldabilidad de circuitos impresos.

- UNE 20-524-77 (II). Técnica de circuitos impresos. Terminología

- UNE 20-531-73. Series de valores nominales para resistencias y condensadores.

- UNE 20-543-85 (I) .Condensadores fijos en equipos electrónicos.

- UNE 20-545-89. Resistencias fijas para equipos electrónicos.

OTRAS:

- UNE 20916: 1995: Estructuras mecánicas para equipos electrónicos. Terminología.

- UNE 21302-2: 1973: Vocabulario electrotécnico. Electrónica de potencia.

- UNE 21302-551: 1996: Vocabulario electrotécnico internacional. Parte 705 propagación de las ondas de radio.

- UNE 21352: 1976: explicación de las cualidades y funcionamiento de equipos de media electrónicos.

- UNE-EN60933: sistemas de audio, video y audiovisuales. Interconexiones y valores de adaptación.

- UNE-EN61000-4-3-1998: Compatibilidad electromagnética.

- UNE-EN61030: 1997: Sistemas de audio, video y audio visuales. Bus digital doméstico.

- EN50090-3-2-1995: Sistemas electrónicos para viviendas y edificios.

- EN123500: 1992: Especificación intermedia: placas de circuitos impresos flexibles con taladros para la inserción de componentes.

5.2. Proceso de fabricación

- **Preparación de componentes**:

Primero se adquieren los componentes teniendo en cuenta sus especificaciones técnicas, a continuación se obtienen las placas de circuito impreso, basándonos en las pautas anteriores. Como último punto, montaje de componentes en placa de circuito impreso y soldadura.

- **Obtención de circuito impreso**:

El circuito impreso es pedido en la página web JLCPCB. Cargamos el archivo gerber en la pagina y esta se encargara de imprimirlo

- **Soldadura y montaje de componentes en placa de circuito impreso**:

Se debe tener muy en cuenta la manipulación de los componentes, ya que este material es susceptible a la hora de su transporte e instalación en circuito impreso. Los dos circuitos integrados de nuestro proyecto deben ser instalados en zócalos, para su instalación, también debemos prever el lugar y la indumentaria del personal de montaje, ya que estos pueden acumular cargas electrostáticas.

5.3. Cláusulas sobre garantías, plazo de ejecución, etc. …

Este tipo de cláusulas intentan proteger a las partes de posibles errores de manipulación del equipo diseñado, así como establecer un período de garantía de funcionamiento del equipo.

Reconocimiento de los materiales.

El cliente queda autorizado a utilizar para el desarrollo de este proyecto los materiales que cumplan las condiciones indicadas en el pliego de condiciones., sin necesidad de reconocimiento previo de la empresa proyectista, siempre y cuando se trate de materiales de procedencia reconocida y suministros normales.

Indemnizaciones por daños y perjuicios.

El cliente no tendrá derecho a indemnización por causas de pérdidas, averías o perjuicios ocasionados en el desarrollo del proyecto.
Será de cuenta de la empresa contratista indemnizar a quien corresponda y cuando a ello hubiere lugar, de todos los daños y perjuicios que puedan causarse por las operaciones de desarrollo y ejecución del proyecto.
El contratista será el responsable de todos los accidentes que sobrevinieran durante la instalación del equipo electrónico, de cualquier avería o accidente.

Plazos de ejecución.

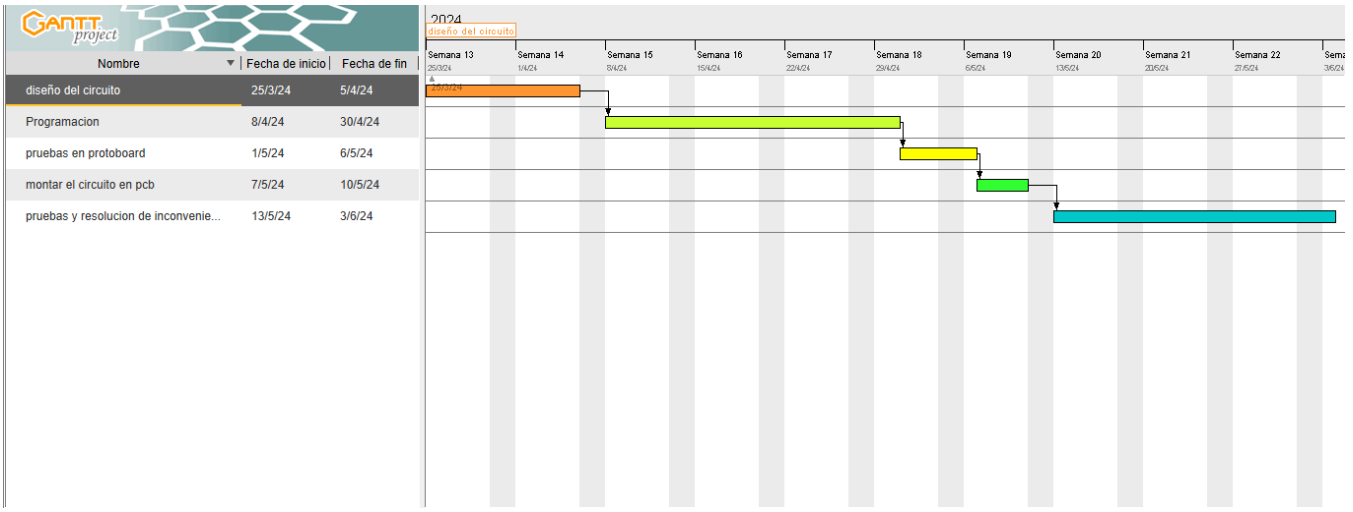
Se indican en el contrato y empezarán a contar partir de la fecha en que se comunique a la empresa proyectista la adjudicación del proyecto.
Los retrasos debidos a causas ajenas a la voluntad de ésta serán motivo de prórroga. El retraso en el pago de cualquier valoración superior a partir de la fecha de la misma, se considerará motivo de prórroga por igual plazo.

Recepción provisional.

Una vez terminado el equipo electrónico en los quince días siguientes a la petición de la empresa proyectista se hará la recepción provisional del equipo por la empresa contratista, requiriendo para ello la presencia de una persona autorizada para cada empresa y levantándose por duplicado el acta correspondiente que firmarán las partes. Si se detectasen fallos de funcionamiento, la empresa contratista lo comunicará por escrito a la empresa proyectista para su reparación fijando un plazo prudencial.

Periodo de garantía.
Como garantía de la bondad de la obra se descontará a la empresa contratista la última liquidación, el 3% del importe total de la obra.

5.3.1. Planificación y Programación (Diagramas de Pert y de Gantt)



5.4. Cláusulas de índole económica.

En estas cláusulas se suele determinar la forma de pago en las distintas fases del proyecto:

Pagos valorados. Mensualmente se hará, entre la empresa proyectista y la contratista, una valoración del proyecto desarrollado, con arreglo a los precios establecidos y con los planos y referencias necesarias para su comprobación. La comprobación y aceptación deberán quedar determinadas en 15 días.

Abonos de materiales. Cuando a juicio de la empresa contratista no exista peligro de hurto de los componentes adquiridos para el desarrollo del equipo electrónico, éstos se abonarán antes de la finalización del proyecto según establezcan las partes, no obstante la empresa contratista podrá exigir las garantías necesarias para evitar la salida o deterioro de los componentes abonados.

Descuento por equipo defectuoso. La empresa contratista podrá proponer a la empresa proyectista la aceptación de estas taras con la rebaja económica que estime oportuna si se ha observado defectos de funcionamiento en el equipo electrónico diseñado con relación a lo exigido en el pliego de condiciones. Si no quedara satisfecho la empresa contratista con la rebaja quedará obligado al rediseño y construcción de toda la parte del equipo electrónico afectada por los efectos señalados.

Revisión de costos. Se revisarán los costos siempre que resulten modificados las condiciones económicas de los costos de materiales en una diferencia superior al 5% al valor prefijado del precio estipulado en el presupuesto.
Cuando la empresa contratista requiera la ampliación de alguna de las especificaciones o características del equipo electrónico se deberá realizar un estudio económico del sobreprecio a pagar por la empresa contratista. De no haber acuerdo, la empresa proyectista quedará relevada del compromiso de ejecución quedando obligada al empresa contratista al abono total de todos los costes de mano de obra, y similares desembolsados hasta el momento por la empresa proyectista.

Abono de obras. Los pagos valorados se abonarán dentro del mes siguiente a la fecha de redacción. Cualquier retraso sobre estos plazos será indemnizado con el interés oficial para efectos comerciales fijado por el Banco de España.

Liquidación definitiva. En el plazo máximo de un mes desde la recepción del equipo electrónico por parte de la empresa contratista ésta deberá realizar la liquidación definitiva. De existir fianza, éste se devolverá en el mes siguiente a la finalización del plazo de garantía estipulado de no haber reclamaciones de terceros por daños, etc.

5.5. Cláusulas de índole legal

En estas cláusulas se delimitan las condiciones en las que ambas partes podrán rescindir el contrato de construcción del equipo electrónico objeto del proyecto.

Modificaciones de obra. El diseño del equipo electrónico podrá ser cambiado total o parcialmente por la empresa contratista, no obstante si la empresa proyectista se considera perjudicada en sus intereses, solicitará la indemnización a que se considere acreedora, y cuya estimación someterán las partes a la decisión de la comisión arbitral. En los casos de suspensión no correrá el plazo.

Derecho de rescisión. La empresa proyectista podrá rescindir el contrato en los siguientes casos:

1. Cuando las variaciones introducidas en el equipo electrónico aumenten o disminuyan el importe total de ésta de un 20%.

2. Cuando por razones ajenas a la empresa proyectista pase más de un años sin poder trabajar en el equipo electrónico.

3. Cuando se retrase más de seis meses el pago de alguno de los pagos valorados estipulados.

Rescisión por incumplimiento del contrato. En el caso de retraso injustificado sobre los plazos fijados se impondrá a la empresa proyectista una multa de 1,5% del presupuesto asignado como pago valorado.

Liquidación en caso de rescisión. Se hará una liquidación única que será la definitiva con arreglo a lo estipulado en este pliego.

Cuestiones no previstas o reclamaciones. Todas las cuestiones que pudieran surgir sobre interpretación, perfeccionamiento y cumplimiento de las condiciones del contrato entre ambas partes serán resueltas por la comisión arbitral.

La comisión arbitral deberá dictar resolución después de oídas las partes dentro de los quince días siguientes al planteamiento del asunto ante la misma. Durante este plazo, la empresa proyectista deberá acatar las órdenes de trabajo indicadas por la empresa contratista sin perjuicio de proclamar las indemnizaciones correspondientes si la resolución le fuese favorable. Entre las resoluciones dictadas por la comisión arbitral figurará en todo caso la proposición en que cada una de las partes deberá participar en el abono de los horarios de las personas que forman la comisión y de los peritos cuyo informe haya sido solicitado por ella.

6. PRESUPUESTO

6.1. Presupuestos parciales

6.1.1. Presupuesto de componentes y material vario

Referencia	Componente	Descripcion	Unidades	precio/unidad	Precio
1	Arduino	Nano	1	7€	7€
2	Pantalla OLED 128x64	SH1106	1	6.95€	6.95€
3	sensor de distancia	VL53L0X	1	4.09€	4.09€
4	sensor de distancia	Sharp 2Y0A21	1	4.92€	4.92€
5	acelerómetro/giroscopio	MPU6050	1	3.93€	3.93€
6	controlador de carga de batería	TP4056	1	1.82€	1.82€
7	codificador rotativo	KY-040	1	1.24€	1.24€
8	conectores JST-XH	4 pines 2.54mm	4	0.15€	0.60€
9	conectores JST-XH	3 pines 2.54mm	2	0.15€	0.30€
10	conectores JST-XH	2 pines 2.54mm	4	0.15€	0.60€
11	Pulsadores	12x12x7.3mm	3	0.68€	2.04€
12	Módulo IR	—	1	8.27€	8.27€
13	Resistencias	10KOhm	4	0.11€	0.44€
14	Resistencia	22K	1	0.23€	0.23€
15	Termoretráctiles	1.0x45mm	4	0.12€	0.48€
16	Termoretráctiles	1.5x45mm	2	0.12€	0.24€
17	cables	22 awg	1	2.00€	2.00€
18	Interruptor deslizante	----	1	1.69€	1.69€
				Subtotal	47€
			IVA	21%	10€
				Total	57€

6.1.2. Presupuesto de Mano de obra

Referencia	Mano de obra	Horas	precio/hora	Precio
1	Mano de obra diseño	20	10€	200€
2	Mano de obra construcción	5	10.00€	50.00€
			Subtotal	250€
		IVA	21%	53€
			Total	303€

6.1.3. Presupuesto de Medios auxiliares e instrumentación

Referencia	Medio Auxiliar/ Instrumento	Unidades	precio/unidad	Precio
1	Estaño	1	8€	8€
2	Flux	1	3.00€	3.00€
3	Estaño	1	4.00€	4.00€
4	Crimpadora	1	10.00€	10.00€
			Subtotal	18€
		IVA	21%	4€
			Total	22€

6.2. Presupuesto general

Referencia	Materiales	Unidades	precio/unidad	Precio
1	Componentes	1	47.00€	47.00€
2	Mano de obra	1	250.00€	250.00€
3	Medios auxiliares e instrumentación	1	18.00€	18.00€
			Subtotal	315€
		IVA	21%	66€
			Total	381€