UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

**Project 1: Environment/Weather Station**

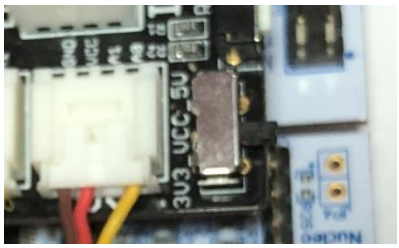
*Jesús Gamero Tello*

*Pedro Millán Álvarez*

01/06/2020

# INDICE

# 1 . Conexionado y sensores utilizados

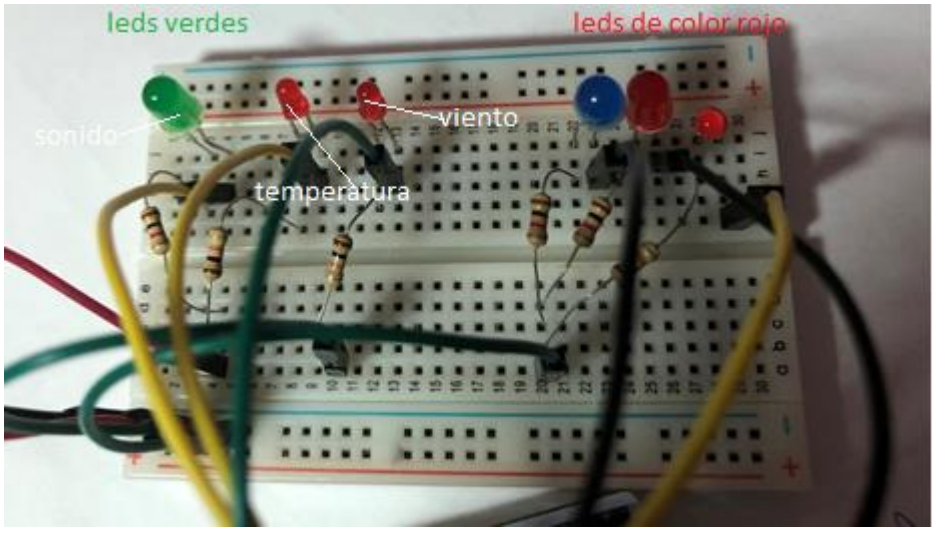



En primer lugar, el proyecto se ha realizado con la placa a 3.3 V como se puede observar en la siguiente imagen, esto es debido a que creo que los datos son más precisos, para ello también se ha tenido que modificar el define MAX_ANGLE de su valor por defecto original que era 4096 a 1024, para realizar los posteriores cálculos de las diferentes fórmulas que nos calculan los diferentes valores obtenidos por los sensores.

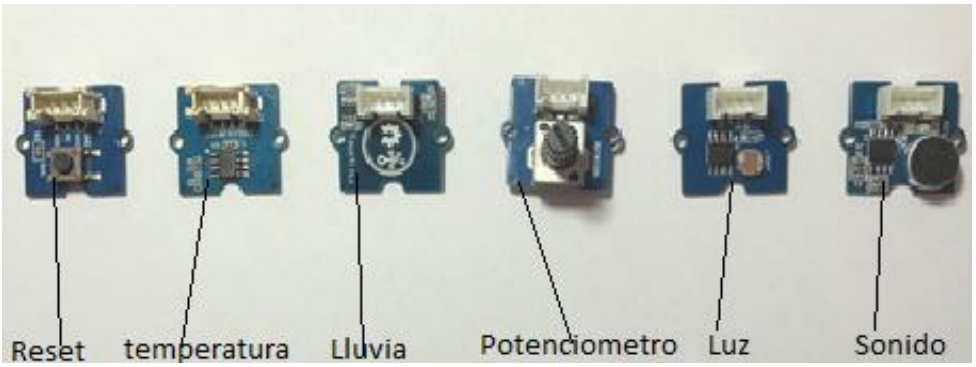

*IMAGEN LCD*

En cuanto al monitor utilizado, hemos optado por usar el LCD, el cual hemos tenido que "puentear" al pin de 5 V para su correcto funcionamiento. En cuanto a los datos mostrados en el LCD decir que la velocidad se mide en **KM/H**, la temperatura en **grados Celsius** el sonido ambiente en **decibelios** y la luz en **lux**, debido a la cantidad de datos que había que mostrar no nos cabían las medidas en el LCD.

En cuanto a los leds se han distribuido en la placa protoboard de manera que los leds de la izquierda son los VERDES y los de la derecha son los ROJOS, debido a que la mayoría de leds de los que se disponían para el montaje estaban fundidos o no lucían se han cogido los que mejor funcionaban.



En cuanto a los sensores y botones utilizados son los especificados en el enunciado de la practica



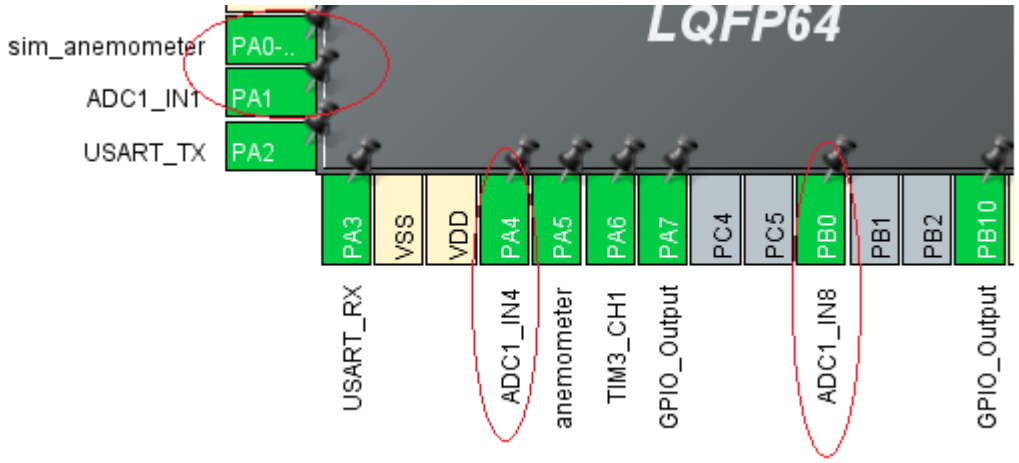

El botón de reset sirve para poner todos los leds a color “verde” es decir el estado inicial. Los sensores de temperatura, luz y sonido son los encargados de medir la temperatura, luminosidad y sonido de la habitación donde se encuentren desplegados. En cuanto al botón táctil servirá para “captar” si llueve (si se pulsa el botón) o no llueve (si no se pulsa el botón). Por último, el potenciómetro nos servirá para simular la velocidad del aire la cual variará entre 0 y 199 Km/h.

## 1.1 Tablas de conexiones

### SENSORES

| Sensor | Puerto | Pin |
|---|---|---|
| Potenciómetro | A0 | PA_0 |
| Sensor de sonido | A1 | PA_1 |
| Sensor de luz | A2 | PA_4 |
| Sensor de temperatura | A3 | PB_0 |

Todos los sensores se han configurado como ADC_IN excepto el potenciómetro cuya configuración ya estaba establecida en la práctica base.



## LEDS

| | Color verde | Color rojo |
|---|---|---|
| **Temperatura** | PB_4 | PB_5 |
| | | |
| **Viento** | PA_7 | PB_6 |
| | | |
| **Sonido** | PA_8 | PB_10 |

Todos los pines se configurarían como GPIO_OUTPUT, a continuación, un ejemplo de los pines PB6, PB5 y PB4


## BOTONES

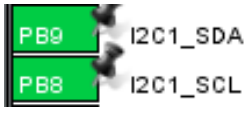
| Botón | Puerto | Pin |
|---|---|---|
| botón Reset | D8 | PA_9 |
| Sensor lluvia | D2 | PA_10 |

Configuraríamos estos pines como GPIO_EXTI a continuación una imagen de como estaría configurado en el STM32CubeMx y activaríamos el NVIC.


## LCD

| | Pin |
|---|---|
| GND | Al pin gnd (ver imagen) |
| VCC | Al pin de 5 V (ver imagen) |
| SDA | PB_9 |
| SCL | PB_8 |

Así es como se quedarían configurados los pines en CubeMx



## Configuración NVIC STM32CubeMx

| | | |
|---|---|---|
| Memory management fault | ☑ | 0 |
| Pre-fetch fault, memory access fault | ☑ | 0 |
| Undefined instruction or illegal state | ☑ | 0 |
| System service call via SWI instruction | ☑ | 0 |
| Debug monitor | ☑ | 0 |
| Pendable request for system service | ☑ | 0 |
| Time base: System tick timer | ☑ | 0 |
| PVD interrupt through EXTI line 16 | ☐ | 0 |
| Flash global interrupt | ☐ | 0 |
| RCC global interrupt | ☐ | 0 |
| ADC1 global interrupt | ☑ | 0 |
| EXTI line[9:5] interrupts | ☑ | 0 |
| TIM2 global interrupt | ☑ | 0 |
| TIM3 global interrupt | ☑ | 0 |
| I2C1 event interrupt | ☑ | 0 |
| I2C1 error interrupt | ☑ | 0 |
| USART2 global interrupt | ☐ | 0 |
| EXTI line[15:10] interrupts | ☑ | 0 |
| DMA2 stream0 global interrupt | ☑ | 0 |
| FPU global interrupt | ☐ | 0 |

## 1.2 Configuraciones adicionales en el código

En cuanto a la parte de los **ADC** en la función *MX_ADC1_Init()* hemos puesto una resolución de 10B

```
 hadc1.Init.Resolution = ADC_RESOLUTION_10B;
```

También se han configurado los diversos canales para posteriormente usarlos mediante DMA y recoger correctamente los datos

```
sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_480CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
```

```
  }
  /** Configure for the selected ADC regular channel its corresponding ra
nk in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = 2;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure for the selected ADC regular channel its corresponding ra
nk in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_4;
  sConfig.Rank = 3;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure for the selected ADC regular channel its corresponding ra
nk in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_8;
  sConfig.Rank = 4;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
```

En cuanto al **TIM3** en la función *MX_TIM3_Init()* hemos configurado el preescaler y el period

```
 htim3.Init.Prescaler = 1119;
htim3.Init.Period = 2999;
```

Y también la Polarity para coger la subida y la bajada de la onda:

```
sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_BOTHEDGE;
```

## 2. LCD

Como hemos hablado al principio, nuestro monitor elegido para mostrar los datos es el LCD.
Nuestro proyecto está realizado con un voltaje de 3.3 V y el LCD necesita 5v para funcionar correctamente, a continuación, explicaremos como realizar la conexión

## 2.1 CONEXIONADO LCD.

EL LCD tiene 4 conexiones:

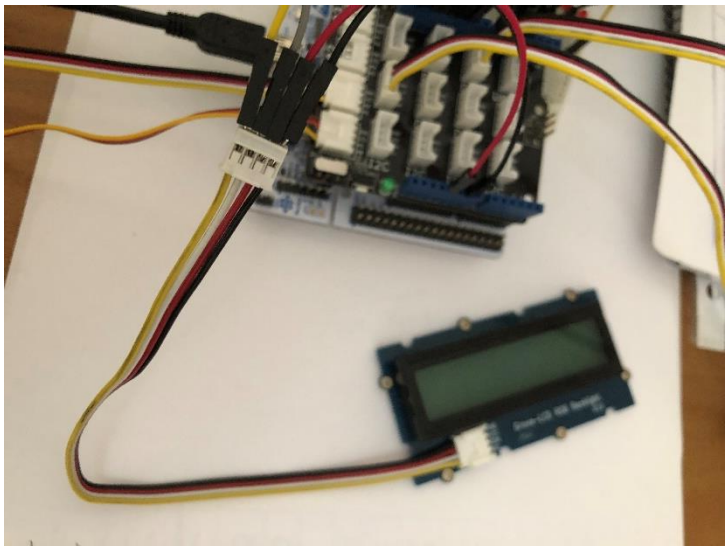
Como vemos esas 4 conexiones son: **GND**(negro) , **VCC**(rojo) , **SDA**(Blanco), **SCL**(amarillo).


Primero conectaremos un cable de 4 hembras como el que se puede observar en la foto anterior.

Una vez conectado el cable hembra de 4 conexiones, necesitaremos 4 cables macho y conectarlos de la siguiente manera al cable hembra.

Una vez tengamos conectado los 4 cables macho, cada uno lo conectaremos tal cual vine escrito en la tabla de conexiones del lcd.

GND(negro) , VCC(rojo) , SDA(Blanco), SCL(amarillo)



## 2.2 FUNCIÓN IMPRIMIR_LCD.

Esa distribución colocada está programada en la función imprimir_lcd.
Con la función **locate(col,row)**, extraída de una librería externa, iremos colocando el cursor para ir administrando la información en el LCD y con la función **HAL_I2C_Master_Transmit();** iremos enviando los datos al LCD.

**Código de la función imprimir_lcd:**

```
void imprimir_Lcd(){
  init();
  volatile char v[7] = "xV:";
  volatile char t[5]="xT:";
  volatile char s[7]="xS:";
  volatile char l[9]="xL:";

  char array[3];
  char arrayT[2];
  char arrayS[3];
  char arrayL[3];
  int numerosV=2;
  int numerosT= 0;
  int numerosS= 2;
  int numerosL=4;
  if (speed<10)
  numerosV=3;
  if (speed>99)
  numerosV = 1;
  if (temperature>=0 && temperature<10)
  numerosT=1;
  if (sound<10)
  numerosS=3;
  if (sound>99)
  numerosS = 1;
  if (brightness<10)
  numerosL=5;
  if (brightness>99)
  numerosL= 3;
  if (brightness>999)
  numerosL=2;
  if (brightness>9999)
  numerosL=1;
  locate(0,0);
  strcat(v, itoa(speed, array, 10));
  HAL_I2C_Master_Transmit(&hi2c1,LCD,v,sizeof(v)-numerosV,100);
```

```
  locate(6,0);
  strcat(t, itoa(temperature, arrayT, 10));
  HAL_I2C_Master_Transmit(&hi2c1,LCD,t,sizeof(t)-numerosT,100);
  locate(11,0);
  strcat(s, itoa(sound, arrayT, 10));
  HAL_I2C_Master_Transmit(&hi2c1,LCD,s,sizeof(s)-numerosS,100);
  locate(0,1);
  strcat(l, itoa(brightness, arrayL, 10));
  HAL_I2C_Master_Transmit(&hi2c1,LCD,l,sizeof(l)-numerosL,100);
  locate(9,1);
  strcat(l, itoa(brightness, arrayL, 10));
  if (rain!=1){
  volatile char p[9]="xLluv:No";
  HAL_I2C_Master_Transmit(&hi2c1,LCD,p,sizeof(p)-1,100);
  }
  else{
  volatile char p[9]="xLluv:Si";
  HAL_I2C_Master_Transmit(&hi2c1,LCD,p,sizeof(p)-1,100);
  rain=0;
  }


}
```

## 3. Casos Utilizados

Se compone de los siguientes casos o estados:

- **Caso 0:** Inicial, los 3 leds están en verde.
- **Caso 1:** La temperatura pasa el umbral de la temperatura.
- **Caso 2:** El sonido sobrepasa el umbral del sonido.
- **Caso 3:** La velocidad del viento sobre pasa el umbral.
- **Caso 4:** La temperatura y sonido sobrepasan ambos sus umbrales establecidos.
- **Caso 5:** La temperatura y la velocidad del viento sobrepasan ambos sus umbrales establecidos.
- **Caso 6:** El sonido y la velocidad del viento sobrepasan ambos sus umbrales establecidos.

- **Caso 7:** Los tres sensores sobrepasan sus umbrales establecidos.

Para realizar estos cambios utilizaremos dos métodos **next_state()** y **encenderLeds()**

# 4. Funciones a comentar

## 4.1 next_state()

Esta función se encargará de decir cuál es el siguiente estado de los descriptos anteriormente.

Se compone de una serie de *if- else if* que comprueban cuales de los sensores han sobrepasado el umbral preestablecido mediante las siguientes variables:

```
#define THRESHOLD_TEMPERATURE 20
#define THRESHOLD_WIND 90
#define THRESHOLD_NOISE 60
```

Según que sensores sobrepasen los umbrales estaremos en un caso diferente de los cuales están descriptos anteriormente.

Código de la función:

```
int next_state(){

  if  (temperature > THRESHOLD_TEMPERATURE && sound <  THRESHOLD_NOISE &&
 speed < THRESHOLD_WIND){
    return 1;
  }
  else if ((temperature < THRESHOLD_TEMPERATURE) && (sound >  THRESHOLD_N
OISE) && (speed < THRESHOLD_WIND)){
   return 2;
  }
  else if (temperature < THRESHOLD_TEMPERATURE && sound < THRESHOLD_NOISE
 && speed> THRESHOLD_WIND){
     return 3;
  }
  else if  (temperature > THRESHOLD_TEMPERATURE && sound >  THRESHOLD_NOI
SE && speed < THRESHOLD_WIND){
    return 4;
  }
  else if(temperature > THRESHOLD_TEMPERATURE && sound <  THRESHOLD_NOISE
 && speed > THRESHOLD_WIND){
```

```
  return 5 ;
  }
  else if ((temperature < THRESHOLD_TEMPERATURE )&& (sound >  THRESHOLD_N
OISE) && (speed > THRESHOLD_WIND)){
    return 6;
  } else if((temperature > THRESHOLD_TEMPERATURE )&& (sound >  THRESHOLD_
NOISE) && (speed > THRESHOLD_WIND)){
      return 7 ;
  }
```

## 4.2 EncenderLeds().

Está función es la que tiene implementado cada uno de los estados para encender los Leds correspondiente a cada caso.

La función empezará en el estado 0, con los 3 leds verdes encendidos (estado al que no se podrá volver en caso de que se pulse el botón correspondiente al D8, que tiene la función de reset).

Dependiendo de que sensor sobrepase el umbral establecido, se irán encendiendo su led rojo y apagando su led verde correspondiente, con la particularidad de que una vez se encienda el led rojo, aunque deje de superar el umbral no se apagará ni se encenderá el verde, únicamente ocurrirá eso en caso de que se pulse el botón correspondiente al D8.

Código de la función:

```
oid encederLeds(int caso)
{
  /* PB4 PA7 PA8 VERDES */
  /* PB5 PB6 PB10 ROJOS */

  switch (caso)
  {
  case 0:
    //reset
    printf("case %i\n", caso);

    GPIOB->ODR |= GPIO_ODR_OD4_Msk;
    GPIOA->ODR |= GPIO_ODR_OD7_Msk;
    GPIOA->ODR |= GPIO_ODR_OD8_Msk;
```

```
  GPIOB->ODR &= ~GPIO_ODR_OD5_Msk;
  GPIOB->ODR &= ~GPIO_ODR_OD6_Msk;
  GPIOB->ODR &= ~GPIO_ODR_OD10_Msk;
  break;

case 1:
  //temperatura se pasa
  //printf("case %i\n", caso);
  GPIOB->ODR |= GPIO_ODR_OD5_Msk;
  GPIOB->ODR &= ~GPIO_ODR_OD4_Msk;
  break;
case 2:
  //sound se pasa
  printf("case %i\n", caso);
  printf("el sound en el case es %d\n",sound);

  GPIOB->ODR |= GPIO_ODR_OD10_Msk;
  GPIOA->ODR &= ~GPIO_ODR_OD8_Msk;
  break;
case 3:
  //Viento se pasa
 // printf("case %i\n", caso);

  GPIOB->ODR |= GPIO_ODR_OD6_Msk;
  GPIOA->ODR &= ~GPIO_ODR_OD7_Msk;
  break;
case 4:
  //temperatura se pasa sound se pasa
  //printf("case %i\n", caso);
  GPIOB->ODR |= GPIO_ODR_OD5_Msk;
  GPIOB->ODR &= ~GPIO_ODR_OD4_Msk;
  GPIOB->ODR |= GPIO_ODR_OD10_Msk;
  GPIOA->ODR &= ~GPIO_ODR_OD8_Msk;
  break;

case 5:
  //Temperatura se pasa viento se pasa
  //printf("case %i\n", caso);
  GPIOB->ODR |= GPIO_ODR_OD5_Msk;
  GPIOB->ODR &= ~GPIO_ODR_OD4_Msk;
  GPIOB->ODR |= GPIO_ODR_OD6_Msk;
  GPIOA->ODR &= ~GPIO_ODR_OD7_Msk;

  break;
case 6:
  //sound se pasa viento se pasa
  //printf("case %i\n", caso);
```

```
    GPIOB->ODR |= GPIO_ODR_OD6_Msk;
    GPIOB->ODR |= GPIO_ODR_OD10_Msk;
    GPIOA->ODR &= ~GPIO_ODR_OD7_Msk;
    GPIOA->ODR &= ~GPIO_ODR_OD8_Msk;
    break;
  case 7:
    //se pasan todos
    //printf("case %i\n", caso);
    GPIOB->ODR |= GPIO_ODR_OD5_Msk;
    GPIOB->ODR &= ~GPIO_ODR_OD4_Msk;
    GPIOB->ODR |= GPIO_ODR_OD10_Msk;
    GPIOA->ODR &= ~GPIO_ODR_OD8_Msk;
    GPIOB->ODR |= GPIO_ODR_OD6_Msk;
    GPIOA->ODR &= ~GPIO_ODR_OD7_Msk;

    break;
  }
}
```

## 4.3 HAL_ADC_ConvCpltCallback()

En esta función propia del ADC usamos un *for* para guardar los valores de cada sensor en un buffer para posteriormente calcular su valor y guardarlo en una variable global.

Los valores se guardaran en las variables : temperature (para la temperatura calculada), brightness (donde se guardara el brillo o luminosidad calculada) , sound (para guardar el valor del sonido ) y speed (donde se guardara la velocidad del viento).

Dentro de cada caso del 0 al 3 calcularemos su valor, mediante la función correspondiente que se explicaran a continuación.

La información de todo lo relacionado con el DMA, lo hemos visto de esta pagina web: https://controllerstech.com/how-to-read-multichannel-adc-in-stm32/

## 4.4 Calcular_viento()

En esta función se calcula el valor de la velocidad del viento, en el cual el valortotal es la diferencia entre la subida y la bajada y el 0.1405, es un valor que sale de realizar 200/(1499-75), siendo 1499 la mitad de la onda

```
void calcular_viento(int i){
    htim2.Instance->CCR1 = 75 + (adc[i]*1425/MAX_ANGLE);
    speed = (valortotal - 75) * 0.1405;

}
```

## 4.5 Calcular_temperatura()

Aquí calcularemos la temperatura, se ha utilizado una formula encontrada en la página seedstudio y se ha aplicado aquí.

La página web: https://wiki.seeedstudio.com/Grove-Temperature_Sensor_V1.2/

```
void calcular_temperatura(int i){
   float R=1023.0/adc[i]-1.0;

    temperature=1/(log(R)/4275+1/298.15)-273.15;
}
```

## 4.6 Calcular_luminosidad()

En esta función se calculará la luminosidad en lux, se ha utilizado una formula encontrada en el foro de Arduino y se han ajustado los valores.

La página web: https://forum.arduino.cc/index.php?topic=331679.0
En la formula de ese foro se divide entre 80, pero nosotros ese valor lo hemos ajustado a nuestro problema, ya que en el foro se especifica que ese valor hay que ajustarlo.
El valor 54.46052 es el valor calculado para ajustar la fórmula de la luminosidad que sale de la siguiente ecuación:
e^(627/x)=100 000
Siendo 627 el valor máximo que recoge el sensor con una alta luminosidad.
X= 627/(log100000/log e)
X= 54.46052

```
void calcular_luminosidad(int i){
  brightness = exp((float)adc[i]/54.46052);
}
```

## 4.7 Calcular_sonido()

Se calculará el sonido ambiente de la habitación, hemos ajustado la formula lo mejor posible para sacar valores lo mas razonables posibles, ya que era muy difícil de ajustar. También se ha buscado en el foro de Arduino para sacar la formula a ajustar

La página web: https://forum.arduino.cc/index.php?topic=534279.0

```
void calcular_sonido(int i){
   if (adc[i]!=0 || adc[i]<0){
```

```
        sound= 20*log(adc[i]/100.0)+50;
  }
}
```

## 5. Video explicando casos YouTube

El enlace al video donde se explican los diferentes casos es:
https://youtu.be/qmFklOSxruM