| Plantilla de Evaluación: (Sólo para profesores) | | |
|---|---|---|
| **Nota Informe (2 puntos)** | **Evaluación Exposición (4 puntos)** | **Nota Proyecto** |
| 60% Nota Final | | |
| | | |

# PROYECTO FINAL: Sonar dinámico

# MICROPROCESADORES

Grado en Ingeniería Electrónica Industrial y Automática
4º Curso, 1º Cuatrimestre
Curso académico 2020-2021

*Autores*

Pablo Dubru Carrasco - 100383146

**DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA**
**Campus de Leganés**
**Avenida del Universidad 30**
**28911 Leganés**

# Contenidos

## 1.1 Objetivos del proyecto

El objetivo de este proyecto es desarrollar un sonar. Un SONAR, acrónimo de Sound Navigation And Ranging, que se basa en una técnica para la navegación por sonido, específicamente con el uso de dispositivos ultrasonidos. Este sonar se configurará para medir distancias en un rango de 180° y devolver el ángulo con la distancia máxima al obstáculo. Para conseguir este propósito se acoplará un sensor ultrasonidos HC-SR04 a un servo motor para habilitar el barrido. El sonar será controlado usando una placa NUCLEO-152RE de Stm32.

Con este proyecto se busca configurar un sistema que sea capaz de detectar la dirección con menos probabilidad de colisión con un obstáculo. Este proceso se debe de poder ejecutar dinámicamente en el caso en el que el módulo se móvil, para asegurar que a lo largo del movimiento no existan colisiones. Cuando la distancia medida sea lo suficientemente grande, se calificará el desplazamiento aceptable en la dirección medida para el dispositivo al que vaya acoplado el módulo. Este tipo de sistema se podría usar en robots móviles como por ejemplo los robots de limpieza automáticos.

Este módulo tendrá dos modos de funcionamiento:

- **Modo 1 –** Este modo es el manual. El usuario podrá posicionar la dirección que medirá el sensor ultrasonidos y medir la distancia al obstáculo más cercano con el uso de un botón en la placa. Una vez medida la distancia se anunciará con un zumbido y se mostrarán la distancia y el ángulo por pantalla.
- **Modo 2 –** Este modo es automático. El módulo realizará automáticamente un barrido de 4 ángulos y medirá la distancia de cada uno de ellos. Una vez completado, el módulo se situará señalando el ángulo con menos probabilidad de colisión. La finalización de este proceso se indicará con un zumbido y una luz led.

Adicionalmente, se implementará una unidad inercial MPU6050 para medir la velocidad angular a la que se mueve el servo motor. Si la velocidad de esta baja por debajo del 50% de su velocidad normal, el programa se detendrá y enviará una señal de error. Aunque este modo no funciona por completo, se mostrará el progreso conseguido y se discutirá maneras de mejorarlo

Por último, se implementará un modo *Stand By*, en el que se mandará al microprocesador a dormir, reduciendo drásticamente el consumo energético de la placa. Este modo se activará como los dos modos mencionados anteriormente, y se despertará con el botón de reinicio.

## 1.2 Descripción del funcionamiento

La interfaz del sistema se compone de un menú principal, en el que se presentarán las opciones disponibles al usuario. Estas serán tres: Modo 1 (Manual), Modo 2 (Automático) y el Modo Stand By. El formato que adopta este menú se puede observar en el minuto 00:12 del video demostrativo. Para poder elegir un modo u otro, se introducirán números individuales, que se son asignados en el menú. Por ejemplo, para seleccionar el Modo 1, se deberá introducir el carácter '1'. Este formato de interacción de la interfaz se mantendrá a lo largo del programa.

Si se selecciona el Modo 1, se pedirá al usuario posicionar el sensor en la posición deseada usando el potenciómetro (Minuto 00:14). Una vez posicionado, se deberá de presionar el botón usuario para ejecutar una medida, devolviendo el programa el ángulo en el que se encuentra el sensor y la distancia al objeto más cercano. Adicionalmente, el zumbador ejecutará una melodía de victoria específica del modo 1 para avisar que la medida ha sido tomada (Minuto 00:30).

Cuando se selecciona el Modo 2 el usuario no deberá de interactuar de ninguna manera con el hardware del sonar. El servomotor automáticamente posicionará sensor ultrasonidos en 4 ángulos predeterminados: 0, 60, 120 y 180 grados (Minuto 00:42). A lo largo de su recorrido, el sensor medirá la distancia al objeto más cercano, guardando esas medidas en la memoria del programa. Una vez que se complete el barrido de medidas, se analizarán las medidas obtenidas del sensor, buscando la mayor de ellas. Una vez encontrada, se devolverá por pantalla junto con la posición en la que fue tomada, posicionando el motor en dicha posición. Al igual que en el modo 1, el zumbador ejecutará una melodía específica para el modo 2 avisando que proceso ha finalizado (Minuto 00:50).

Una vez que el funcionamiento del modo elegido ha finalizado, al final tanto del Modo 1 como el 2 se preguntará al usuario si desea volver a ejecutar el modo seleccionado o regresar al menú principal para ejecutar otro problema. Para volver a medir se introducirá '1' y para salir '0' (Minuto 00:52).

Si se selecciona el Modo *Stand By*, la placa se dormirá y entrará en mode stand by. Durante este modo el consumo energético de la placa será mínimo, y no se podrá interactuar con ella, incluso si se le mandan valores por puerto serie (Minuto 01:00). Para volver a despertarla, se usará el botón de reinicio colocado en la placa. Para asegurarse que la placa esta despertando del modo Stand By, y no solo reiniciándose, se mandará un mensaje especificando que ha despertado del modo stand by. Este mensaje solo se mostrará si la placa ha entrado anteriormente en el modo Stand By.

Por último, está el funcionamiento del dispositivo MPU6050. Aunque no estará disponible para visualización del usuario, se adjuntará un programa en el que se podrá visualizar los valores que toman acelerómetro y el giroscopio del dispositivo (Minuto 01:09). Esto permitirá analizar como varían los valores dependiendo del eje y de la velocidad a la que se giren. El acelerómetro proporcionará valores en g, que es la constante gravitacional (aceleración). Por otro lado, el giroscopio mostrará valores en grados por segundo (velocidad).

Estos valores se usarán para analizar la velocidad a la que se mueve sonar, y se utilizarán únicamente en el modo automático. Cuando se mande una señal de variar la posición del servomotor, se tomará una muestra de la velocidad a la que gira el sonar. Si esta esta por debajo de 100 grados por segundo, se comunicará que existe un obstáculo obstruyendo el giro del sonar.

## 1.3 Descripción de la solución hardware

### 1.3.1 Componentes externos al microprocesador

Para crear el sonar se utiliza una serie de componentes hardware que interactúan entre ellos para conseguir el propósito explicado al comienzo del informe. Un diagrama del montaje del dispositivo se muestra en el Anexo 1. La lista de estos componentes es la siguiente:

#### Micro Servo Tower Pro SG90 9g

Este servo motor se usa para girar sensor ultrasonidos, así pudiendo medir distancias en un barrido de 180 grados. Este tipo de servo motor se controla mediante señales PWM, en las que dependiendo del '*duty cycle*' de la señal, el sensor toma una posición específica (Anexo 2). Para ello se habilita un timer generando dicha PWM con la siguiente configuración:

$$PSC = 16 - 1$$

$$ARR = \frac{t * f_{CLK}}{P} - 1 = 3999$$

Para poder controlar el motor por lo tanto se modifica la duración de la señal PWM a través del registro CCR. El valor que toma en el código se explica más adelante en la práctica.

#### Sensor ultrasonidos HC-SR04

Este es el sensor de ultrasonidos que mide la distancia al objeto más próximo del mismo. Este tipo de sensores tienen 4 pines: Vdd, GRND, Trigger y Echo (Anexo 3). El funcionamiento del sensor es el siguiente: se debe enviar una señal de 10 microsegundos al pin Trigger del sensor, que activará el sensor hará que mande 8 pulsos de ultrasonidos (Anexo 4). Desde el momento que se envían estos pulsos, el pin Eco se activa, manteniéndose así hasta que el sensor reciba los pulsos devuelta una vez que rebotan contra el objeto, que vuelve a desactivarse. Durante este tiempo, se activa un temporizador para medir el tiempo que pasa entre ambos sucesos. Dividiendo el tiempo obtenido entre 58, se obtiene la distancia al objeto.

Para que el proceso explicado anteriormente funcione se necesitará activar un pin de salida (GPIO_Output) para poder mandar la señal de 10 microsegundos al Trigger y un timer en modo '*Interrupt Capture Mode*', que habilita una interrupción cada vez que detecta un flanco de subida o bajada. La configuración final del timer es la siguiente:

PSC: 31999 (us)

ARR: 65535 (max)

Mode: Interrupt capture mode

Polarity selection: Both edges

Global interrupt enabled

#### Zumbador piezo-eléctrico

Este componente es un zumbador pasivo que genera sonidos cuando le introduces una señal PWM. Se compone por dos pines, uno conectado a tierra y el otro a la señal PWM entrante. El volumen y el tono del sonido dependen de dos características de la señal entrante: el duty cycle y la frecuencia. Al incrementar el valor del duty cycle, el volumen del sonido generado incrementará. Por otro lado, si la frecuencia de la señal PWM incrementara, el sonido que emite el zumbador será más agudo.

Por lo tanto, para poder generar canciones con múltiples notas musicales, será necesario configurar un timer con una configuración dinámica, que varíe para poder generar la frecuencia deseada para cada nota musical. Para ello se mantendrá el valor del ARR fijo a 256, mientras que se varía PSC usando la siguiente fórmula:

$$f_{PWM} = \frac{32MHz}{(ARR+1)(PSC+1)}$$

Con esa fórmula se podrá ajustar la frecuencia de la señal entrante y consecuentemente el tono de la nota. La configuración de la frecuencia se discutirá en el apartado de software.

#### Potenciómetro

El potenciómetro es una resistencia variable que se usa como regulador de la posición del servomotor en el Modo 1. Usando el ADC incorporado en el microprocesador, es posible representar la resistencia del potenciómetro como un número de 0 a 4095. El circuito eléctrico del potenciómetro se puede ver en el Anexo 5.

Si se ajusta el valor, se puede por lo tanto usar para regular el duty cycle de la señal PWM del motor y consecuentemente controlar su posición. Para ello se habilita IN0 como ADC, manteniendo la configuración predeterminada que proporciona el IDE.

#### Dispositivo MPU 6050

Este dispositivo se usa para medir velocidad y aceleración de un cuerpo en movimiento, y se usará con el fin de evitar que el sonar se encuentre con un objeto que le impida el movimiento. Este tipo de dispositivos usan I2C como protocolo de comunicación, por lo que se activan dos pines con este protocolo de comunicación, uno para el reloj (SCL) y otro para los datos (SDA).

Para el control y la obtención de datos se usa la librería TJ_MPU6050, que se explicará con más detenimiento en el apartado de software.

### 1.3.2 Recursos internos del microprocesador

#### Botón usuario

Este botón es usado en el Modo 1 para realizar una medida de distancia con el sensor de ultrasonidos. Se configura como una interrupción (EXTI13)

#### Botón reinicio

Este botón se usa para despertar el microprocesador cuando se encuentre en el modo Stand By. Es por ello que se configura como pin del '*Wakeup call*' (PC13-WKUP2).

#### Universal Asynochronous reciever-trasnmiter (USART)

Se activa internamente el microchip USART para funcionar en modo Asíncrono. Al activar estos pines se permite que la placa envíe y reciba información de una terminal. Se configura con una tasa de baudios de 115200 Bits/s.

### 1.3.3 Tabla de recursos

Una vez mencionados todos los recursos usados en el proyecto, se recopilan todos en la tabla de la Figura 1. Se puede encontrar en el Anexo 6 una foto de la organización de pines.

| Recurso | Pin | Función | Configuración | Intern |
|---|---|---|---|---|
| USART2 | PA2/PA3 | Recibir y enviar datos a una terminal | Baud rate: 115200 Bits/s<br>Word length: 8 bits | No |
| ADC | PA0 | ADC para la señal del potenciómetro | Modo: Asynchronous clock | Sí |
| Timer 2 | PB10 | Señal PWM para el posicionamiento del servomotor | Modo: PWM generator<br>PSC: 15<br>ARR: 39999 | No |
| Botón usuario | PC13 | Botón usuario para toma de medidas en Modo 1 | GPIO_EXTI13 | Sí |
| Botón reinicio | PC13 | Botón de reinicio para despertar el microprocesador | PC13_WKUP2 | Sí |
| Protocolo comunicación I2C | PB9/PB8 | Pines de comunicación con el MPU6050 | Configuración predeterminada | No |
| Timer 4 | PB6 | Señal PWM para el zumbador | PWM Generator<br>PSC: Variable<br>ARR: 255 | No |
| Pin Trigger | PB5 | Señal de 10 microsegundos del trigger para el ultrasonidos | GPIO_Output | Si |
| Timer 3 | PB4 | Timer para detectar flancos de subida y bajada en el pin Echo | PSC: 31999 (us)<br>ARR: 65535 (max)<br>Modo: Interrupt capture mode<br>Polarity selection: Both edges<br>Global interrupt enabled | No |
| Timer 7 | - | Timer para generar un delay de us | PSC: 31<br>ARR: 65535 | No |

*Figura 1 – Tabla de recursos*

## 1.4 Descripción de la solución software

### 1.4.1 Descripción del programa principal

#### Navegación entre bloques principales

El programa principal se divide en 4 bloques: el menú principal, donde el usuario elige el modo a ejecutar, el Modo 1, el Modo 2 y el Modo Stand By, todos explicados en el primer apartado del informe. El diagrama de flujo de estos bloques se puede ver en el Anexo 7.

Para enviar la información a la terminal y que se imprima se hace uso de un macro que ejecuta la línea de código '*HAL_UART_TRANSMIT*' (Anexo 8), que se encarga en enviar una línea de datos por la UART a la terminal.

De la misma manera que se debe de enviar datos a la terminal, el programa debe de poder recibir información que el usuario envíe. Para ello se implementa una función llamada '*readMsg*', mostrada en el Anexo 9. En esta función se lee los datos introducidos por el usuario, devolviendo un array con estos y posteriormente analizados por el programa para analizar la petición. Aunque se puede enviar y recibir datos por terminal con simples líneas de código a lo largo del programa, la creación de las funciones agiliza y reduce la cantidad de código usada.

#### Bloque 1 – Menú principal

En el menú se muestra la lista de modos disponibles y el punto de partida de la interfaz. Como se ha explicado en anteriores apartados, el usuario interactúa con el sistema operativo introduciendo números individuales. Por lo tanto, para pasar de un bloque a otro, se utilizan '*if statements*' para analizar el valor introducido. En este caso, se analiza el valor ASCII de la entrada. Un ejemplo sería la comparación para entrar al Modo 1, donde se busca que el valor introducido sea 49 en vez de 1.

#### Bloque 2 – Modo 1

Este modo habilita el movimiento del servo motor con el potenciómetro hasta que se presiona el botón usuario, que es cuando se mide la distancia y se muestra por pantalla. El diagrama de flujo de este bloque se puede ver en el Anexo 10.

En cuanto al código de este modo, se aprecia como el programa se encuentra en un bucle que habilita el movimiento del servomotor hasta que la se presiona el botón, se activa la interrupción, y se activa la variable '*msg*' para que el sensor ultrasonidos mida la distancia. Todo el código de los periféricos e interrupciones se explicarán más adelante.

Una vez se mide la distancia, el programa devolverá por pantalla tanto el ángulo del sensor como la distancia al objeto más próximo con las funciones mencionadas anteriormente. Adicionalmente se activará el zumbador. Al final se pregunta al usuario si desea volver a realizar una medida de distancia con la función '*readMsg*'.

#### Bloque 3 – Modo 2

El diagrama de flujo de este modo se puede ver en el Anexo 11. Al activar este modo se entra automáticamente en un '*for loop*', que se repetirá el mismo número de veces que de ángulos a analizar. El código del bucle en cuestión se encuentra en el Anexo 12.

Dentro de un bucle se ejecutan los siguientes pasos: primero se obtiene de un array predeterminado (*CCR_Motor[]*) el valor que debe de tomar el registro de la señal PWM del motor, que lo posicionará en el ángulo deseado. Después se ejecutará el análisis de la velocidad del sonar y por último se activará el sensor ultrasonidos para que mida la distancia. Una vez se completa este proceso, se almacena en un array (*distances[]*) la distancia medida.

Una vez se completan todas las iteraciones de medidas, se pasa a analizar estas para encontrar el ángulo con menor probabilidad de colisión. Para ello se ejecuta un código de análisis, donde se puede ver el diagrama de flujo en el Anexo 13 y el código en el Anexo 14. Se basa en un bucle en el que se va comparando los valores del array de distancias. Inicialmente, el primer valor del array es el mayor, y si alguno de los siguientes es mayor, se guarda, y se mantiene hasta alguno mayor tome su lugar o se acabe el bucle. Al acabar se devuelve la mayor medida.

Al mismo tiempo que existe un array predeterminado para posicionar el motor, también se incluye otro con los respectivos ángulos, ambos colocados en las mismas posiciones. Al tener también las medidas organizadas en las mismas posiciones, solo es necesario registrar la posición en la que se encontraba la medida mayor. Así obtenemos la distancia y ángulo del camino con menos probabilidad de colisión. Una vez que se devuelven los datos, se ejecuta la segunda canción para que suena el zumbador y se pregunta si el usuario desea hacer otra medida o volver al menú principal.

#### Bloque 3 – Modo Stand By

En este modo se habilita una función llamada '*gotoSleep'*, donde se habilitan los registros necesarios para que el microprocesador entre en el Modo Stand By. Dicha función se puede ver en el Anexo 15. Para poder ejecutar este modo es necesario activar el '*Power Control Clock*', y seleccionar el modo '*Deep Sleep*' en el '*Cortex System Control Register*', ya que el modo stand by es un subtipo del modo '*Deep Sleep*'. Una vez se habilitan los pines mencionados, se debe de borrar el '*Wake up flag*', ya que si está activada la placa se despertará inmediatamente después, y activar el pin del '*Wake up call*' para que sea posible despertarla. En nuestro caso se activará el pin 2, que se corresponde al botón de reinicio. Por último, se activa el '*Request for interrupt*', que esperará a que el botón de reinicio sea presionado para despertar.

Para confirmar que cuando se presiona el botón de reinicio el microprocesador sale del modo stand by y no se resetea normalmente, se introduce al principio del programa una condición. Esta condición comprueba en el '*Control Status Register*' si la flag del modo Stand By está activada o no. Si lo está, se imprime una frase indicando que está saliendo del modo Stand By.

### 1.4.2 Descripción de las rutinas de atención de interrupción

A lo largo del programa solo se hace uso de 1 caso de interrupción, y este es en el Modo 1, cuando se presiona el botón usuario para realizar la medida. El código de esta interrupción se basa en enviar un mensaje externo al código principal de que el botón ha sido presionado y que comience el sensor a medir distancia. Una vez se envía ese mensaje, se reinicia la flag de la interrupción.

### 1.4.3 Programación de los periféricos

#### Servomotor

El control del servomotor es simple. Es necesario varía el registro CCR, que varia el periodo de la onda PWM y consecuentemente varía la posición del servomotor. Para poder calcular el rango de valores que se le puede introducir al motor se puede usar la siguiente fórmula:

$$Ciclos\ de\ reloj = t * 2MHz$$

El rango obtenido es entre 800 y 2500, aunque estos valores no se ajustan bien al servo motor, ya que con 800 tiembla y tiene problemas para responder correctamente, y con 2500 no llega a hacer el barrido de 180 grados. Después de varias pruebas, se concluye que el rango bajo el que funciona el servo motor es entre 1200 y 5000, siendo el primero 0 grados de ángulo y 180 para el segundo.

### Sensor ultrasonidos

El código para el funcionamiento del sensor de ultrasonidos se puede dividir en tres partes: la señal trigger de 10 microsegundos, la medición de tiempo del pin echo y el calculo de la distancia con dicho tiempo. El diagrama de flujo de la medida con el sensor ultrasonidos se puede ver en el Anexo 16.

Para poder generar la señal de 10 microsegundos, es necesario habilitar una base de tiempos en el timer 7 para poder generar un delay del orden de microsegundos. Para ello se crea una función llamada *'delay'*. Una vez creada esa función, se usa para enviar una señal a través del pin 5 durante 10 microsegundos, activando así la ráfaga de 8 pulsos de ultrasonidos. El código de esta señal se puede observar en el Anexo 12.

Una vez se ha activado el sensor, se pasa al código situado en la interrupción del Timer 3. Este código se ejecutará cuando en el pin del Echo se detecte un flanco de subida o de bajada. Estos flancos ocurren cuando se lanza la ráfaga de pulsos (flanco de subida) y cuando recibe la ráfaga de vuelta (flanco de bajada). La lógica es simple: se borra inicialmente la flag de la interrupción y se pasa a analizar el valor del pin del Echo. Si esta activado, implica que se acaba de lanzar la ráfaga de pulsos, por lo tanto, reiniciando la el timer 3. Por otro lado, si está desactivado implica que las ráfagas han vuelto y se debe de comprobar el tiempo que ha transcurrido. Después de guardar el tiempo, se indica que se ha terminado de medir el tiempo transcurrido con la variable *time_captured* y se envía al código principal el tiempo. El código de la interrupción y el diagrama de flujo se pueden observar en el Anexo 17.

Una vez que se ha obtenido el tiempo transcurrido, se divide entre 58 para obtener la distancia el sensor al objeto.

### Zumbador

Como se menciona anteriormente, el zumbador funciona con señales PWM que, dependiendo de sus características, produce un sonido determinado. Para poder generar canciones específicas, se modificará el PSC del timer que genera dicha señal PWM usando la siguiente fórmula:

$$f_{PWM} = \frac{32MHz}{(ARR+1)(PSC+1)}$$

Usando una página web, se obtiene una serie de notas musicales que constituyen dos canciones, una para anunciar el final de cada modo. Una vez se tienen las notas musicales, se busca en internet también la frecuencia de cada una, y con la formula mostrada arriba, se calcula el valor de PSC necesario. También se ajusta el volumen al que sonará el zumbador variando el valor del registro CCR.

Por último, es necesario ajustar el tempo de la canción, por lo que se ajusta un delay de milisegundos entre las notas. A mayor el tempo, menor el delay entre las notas.

Una vez se escriben las líneas de todos los cambios de notas, se introducen en funciones para poder ser reutilizadas a lo largo del código de manera más cómoda. El código de una de las canciones se puede ver en el Anexo 18.

### MPU6050

Es dispositivo se encarga en medir cambios de velocidad y aceleración en los ejes x,y,z. Para poder controlarlo se hace uso de la librería TJ_MPU6050. De esta librería se busca obtener los valores escalados de giroscopio con unidades de grados por segundo y los valores escalados del acelerómetro con unidades de g.

Para hacer uso del MPU6050, es necesario primero inicializarlo con la función *MPU6050_Init*, que toma como argumento un puntero a los puerto I2C con los que se comunicaría con el microprocesador. Siguiente es necesario configurar el dispositivo, por lo que se usa la función *MPU6050_Config*, que toma como argumento un puntero de tipo *MPU_ConfigTypeDef*. Este objeto es una estructura que configura el funcionamiento del dispositivo. Es en esta estructura se definen los siguientes parámetros y se pueden ver en el Anexo 19:

- Accel_Full_Scale – Se define el rango de medida que tiene el acelerómetro. Para este proyecto, se configura con un rango de +-2g
- ClockSource – Se define la configuración del reloj con el que funciona el dispositivo MPU
- Digital Low Pass Filter (DLPF) – Se define el filtro de paso bajo con el que se filtra los datos obtenidos
- Gyro_Full_Scale – Se define el rango de medida que tiene el giroscopio. Teniendo el cuenta que el servomotor tarda alrededor de 100ms en hacer la rotación (Según el datasheet) se configura el rango a un máximo de 500 grados por segundo
- Sleep_mode_bit – Bit para determinar si el dispositivo esta activo o no. Se define inicialmente a 1 para que este desactivado, ya que solo se usará en el Modo 2. Una vez que se entre en el Modo 2, se activará dicho bit para despertarlo, así ahorrando energía.

Una vez se configura el dispositivo, se debe pasar las variables de las estructuras de la librería del giroscopio y del acelerómetro a variables locales en el proyecto. Para ello se crean dos variables de tipo *ScaledData_Def*, una siendo *myAccelScaled* y la otra *myGyroScaled*. Ambas variables contienen floats con los valores de aceleración y velocidad de los tres ejes.

Por último, se deben usar las funciones *MPU_Get_Accel_Scaled* y *MPU_Get_Gyro_Scaled* para obtener los valores escalados en los 3 ejes. Una vez que se obtienen el siguiente paso es analizar los valores obtenidos para determinar el movimiento del sonar.

Una vez que se activa el modo 2, se debe de poner el bit de sleep mode a 0 para activar el dispositivo. Una vez despierto se debe de analizar cuando se debe de ejecutar las funciones *Get*. Teniendo en cuenta que los tramos que el modo automático recorre son todos de 60 grados, se concluye que debe de tardar unos 33 milisegundos en completarlos, asumiendo que recorre 180 grados en 100 milisegundos. Sabiendo esto, se tomarán 4 muestras a lo largo del recorrido, uno cada 8 milisegundos, por lo q se incluye un delay antes de cada función *Get* del MPU.

Una vez se ha hecho el delay se debe de analizar los datos obtenidos. El array de datos obtenidos debería de tener 2 valores a 0, y dos por encima de 250 grados por segundo aproximadamente. Por lo tanto, se escanea que al menos uno de los datos este por encima de 250. Si no lo hay, implica que existe un obstáculo obstruyendo el paso y existe un error. El código de esta lógica se puede ver en el Anexo 12, aunque en el proyecto final se comente, ya que no funciona correctamente. En la conclusión se analizarán las razones por las que este sistema no funciona.

### Potenciómetro

El potenciómetro tiene como función regular a través de un ADC la posición del servomotor en el Modo 1. El proceso que sigue es simple: primero se debe de habilitar tanto el ADC y el temporizador que genera la señal PWM para controlar el motor. Después se espera a que el ADC haga la conversión entre la señal analógica a la señal digital, y una vez es completada pues se registra ese valor a una variable local. Esta después se igualará al registro CCR del servo motor para posicionarlo.

Un problema que ocurre es que la salida digital del ADC es un valor numérico entre 0 y 4095, mientras que para controlar el servomotor en todo su rango el valor a introducir debe de estar entre 1200 y 5000. Es por ello por lo que se necesita hacer una escala para obtener los valores correctos, por lo que se usa la siguiente fórmula:

$$Registro\ CCR = \left(valor_{adc} * \frac{3800}{4095}\right) + 1200$$

Para el cálculo del ángulo se aplica otra fórmula similar:

$$Ángulo = valor_{adc} * \frac{180}{4095}$$

## 1.5 Estudio energético

Se realizarán tres estudios energéticos, uno por cada Modo del programa. Estos estudios mostrarán el consumo eléctrico del sistema a lo largo de su funcionamiento. Se analizarán picos de consumo si es posible reducir el consumo medio de cada modo.

### 1.5.1 Consumo Modo 1

Observando la primera gráfica, mostrada en el Anexo 20, el modo 1 consume una media de 8.76mA, teniendo un pico de consumo de 9.24mA y una mínima de 7.75mA. El consumo máximo ocurre cuando se permite al usuario mover el servo motor usando el potenciómetro, donde se tiene activado el ADC y el TIMER 2. El mínimo por otro lado es cuando se activa la interrupción con el botón usuario, ya que se usa únicamente GPIOC.

Analizando los resultados, se concluye que el componente que más consume es el ADC, causando el pico de consumo. Por otro lado, la subida de consumo generada por el sensor ultrasonidos (8.21mA) es muy baja, estando 0.3mA por debajo de la media de consumo. El consumo de este modo depende adicionalmente del tiempo que el usuario tarde en posicionar el servomotor. Sumándole al hecho que el movimiento del servo es la etapa que más consume, el consumo de este modo puede ser significativo.

Aun así, los componentes activos en cada etapa no se pueden modificar, ya que se pondría en compromiso el funcionamiento básico del programa.

### 1.5.2 Consumo Modo 2

Observando la siguiente gráfica, mostrada en el Anexo 21, el modo 2 consume una media de 7.96mA, 1mA menos que el modo 1, recalcando como los pines habilitados para el ADC consumen energía. Se pueden observar que los picos de mayor consumo ocurren cuando se realiza la medida de sensor de ultrasonidos, siendo de 8.44mA. Con el consumo mínimo ocurre cuando se envían datos por puerto serie a la terminal, teniendo la USART activada.

Si comparamos el proceso de sensor ultrasonidos en ambos modos, se aprecia que en el modo 2 se consume 0.23mA de más. Este consumo se atribuye al dispositivo MPU6050, que se encuentra activo durante todo el proceso. Adicionalmente, el consumo al mover el servo motor es también notablemente menor, ya que no se necesita alimentar el ADC.

Para poder disminuir un poco el consumo, se podría poner el dispositivo MPU6050 en modo sleep durante el proceso de medida del sensor y volver a encenderlo antes de mover el servomotor.

### 1.5.3 Consumo Modo Stand By

Observando la última gráfica, mostrada en el Anexo 22, el modo 3 consume una media de 1.34mA, un valor mucho menor que los modos 1 y 2. El punto de mayor consumo es cuando se envían datos por puerto serie con 7.79mA de consumo, y un punto mínimo cuando la placa se encuentra en modo Stand By, con un consumo de 290nA.

En este caso, la media de consumo de este modo depende en el tiempo que se encuentre el microprocesador en modo Stand By, ya que el consumo se reduce un 99.6% con respecto al consumo base de este. Se puede apreciar como la activación de esto modo muestra un ahorro de consumo muy notable.

### 1.5.4 Batería elegida

Observando que el consumo máximo que tiene el programa es en el Modo 1 con 9.24mA de corriente, se decide de incluir la batería Li-SOCL2(AAA700), teniendo una capacidad máxima de corriente de 10mA. Esto disminuirá el precio de las baterías. Se decide de poner 2 en paralelo, para pasar la duración de estas de 3 a 7 días.

## 1.6 Conclusión

El dispositivo es capaz de alcanzar su funcionalidad básica, que es analizar un rango de ángulos automáticamente para encontrar el camino con menos resistencia. La precisión de las medidas es suficientemente alta para que sean seguros para un dispositivo móvil pueda usarlo. Aun así, el sistema tiene muchas áreas donde se podría mejorar.

El número de ángulos que el sonar cubre es limitado, pudiendo darse el caso que el camino con menos resistencia no se encuentre en ninguno de ellos. Aunque no es un error determinante en el funcionamiento del sistema, puede llevar a que el dispositivo móvil tenga que usar el sonar múltiples veces en vez de detectar el camino correcto a la primera. Esto se puede solucionar usando pasos adaptativos para escanear los 180 grados frente al sensor en vez de usar posiciones predeterminadas.

Por otro lado, aunque el modo stand by es capaz de reducir drásticamente el consumo energético del sistema, no es un modo como del que entrar y salir. Una manera por lo tanto de reducir aun más el consumo es activar el modo low power mientras el sistema espera que el usuario interactúe con la interfaz. Esto permitiría conseguir la reducción de consumo al mismo tiempo que no se pone en compromiso el funcionamiento de ningún periférico, ya que durante esta espera no se activa nada.

Por último, se debería de discutir posibles razones por las que el dispositivo MPU6050 no es capaz de medir los valores deseados. La razón por la que no es posible usar la lógica explicada en apartados anteriores es debido al enorme error presente cuando se recopilan los datos tanto del acelerómetro como del giroscopio. Estos fluctúan tanto que no es posible crear una condición fiable con la que analizar la velocidad del sonar, llevando a errores falsos y bloqueos no detectados. Esto se puede deber a que el dispositivo esta acoplado a una estructura endeble y que sufre inclinaciones y perturbaciones constantemente, llevando cambios bruscos y no deseados en los valores que devuelve. Para poder mejorar este error se debería de construir un esqueleto mucho más firme para acoplar todos los dispositivos y donde el MPU6050 pueda estar completamente plano, sin inclinaciones indeseadas.

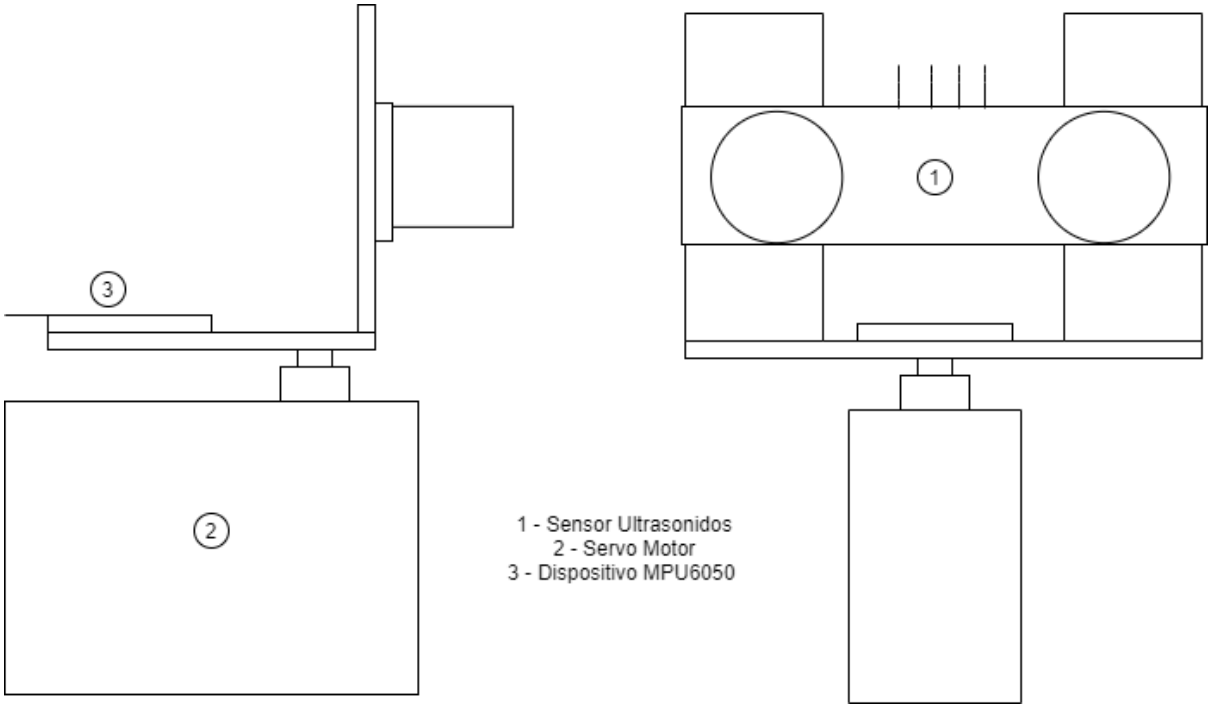
## 1.7 Relación de componentes y valoración económica

| Nombre componente | Precio | Enlace |
|---|---|---|
| Servo Motor Tower pro sg90 | 1.38€ x Unidad | https://www.amazon.es/ALLOMN-Helic%C3%B3ptero-Ambici%C3%B3n-Veh%C3%ADculo-Autom%C3%B3vil/dp/B07Q1GJJZS |
| Sensor ultrasonidos HC-SR04 | 2€ x Unidad | https://www.amazon.es/ELEGOO-Ultrasonidos-Distancia-Raspberry-Disponible/dp/B06W9JD4X2 |
| MPU6050 | 6.5€ x Unidad | https://www.amazon.es/SODIAL-MPU-6050-giroscopio-Acelerometro-Arduino/dp/B00K67X810 |
| Potenciómetro de 10KΩ | 0.99€ x Unidad | https://solectroshop.com/es/potenciometros/486-potenciometro-b100k-lineal.html |
| Zumbador piezo eléctrico 90dB | 1.77€ x Unidad | https://es.rs-online.com/web/p/componentes-de-piezo-buzzer/0238088/ |
| Placa NUCLEO-L152RE | 15€ x Unidad | https://os.mbed.com/platforms/ST-Nucleo-L152RE/ |
| Cables Dupont Macho-Macho + Hembra-Macho | 7€ el conjunto | - |
| **PRECIO TOTAL:** | 34.64€ | |

Aunque se habla sobre una batería externa en el estudio energético, al usar como fuente de alimentación el puerto USB del ordenador, no se incluirá el precio de las baterías en la tabla de componentes.

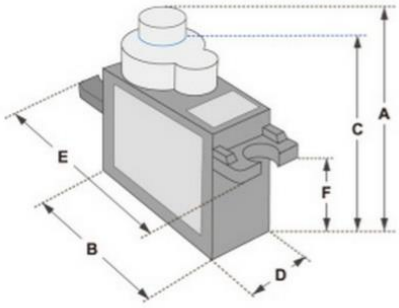
## 1.8 Anexos

### Anexo 1



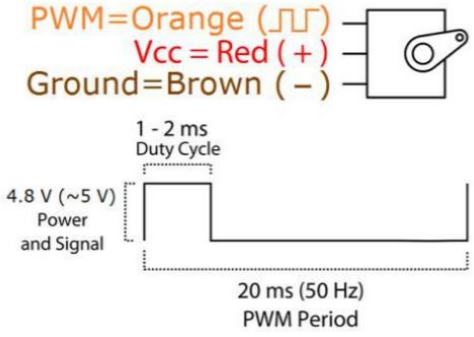

*Anexo 1 – Montage del sonar*

### Anexo 2




| Dimensions & Specifications |
|---|
| A (mm) : 32 |
| B (mm) : 23 |
| C (mm) : 28.5 |
| D (mm) : 12 |
| E (mm) : 32 |
| F (mm) : 19.5 |
| Speed (sec) : 0.1 |
| Torque (kg-cm) : 2.5 |
| Weight (g) : 14.7 |
| Voltage : 4.8 - 6 |

Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is middle, is all the way to the right, "-90" (~1ms pulse) is all the way to the left.

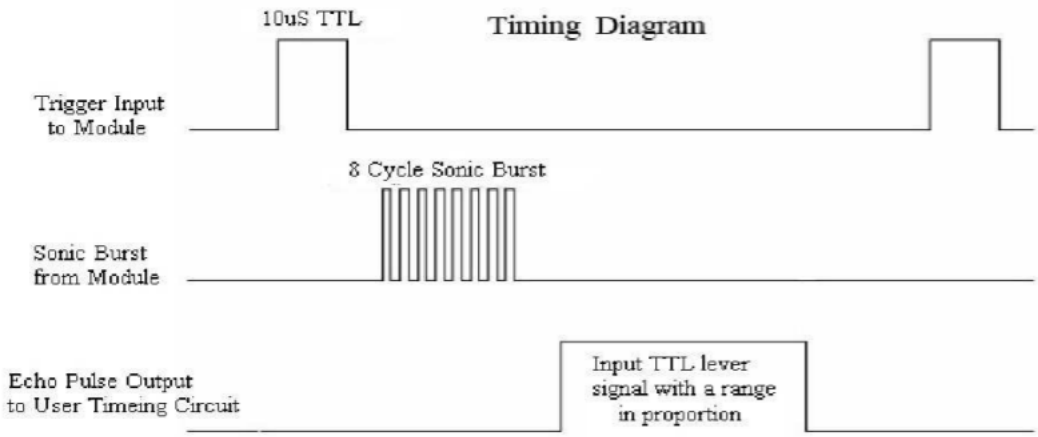



*Anexo 2 – Datasheet servomotor*

## Anexo 3

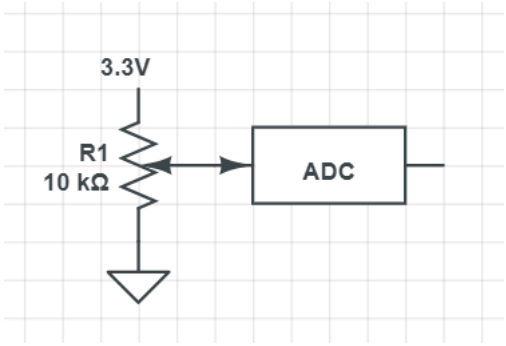

*Anexo 3 – Pines del sensor ultrasonidos*

## Anexo 4



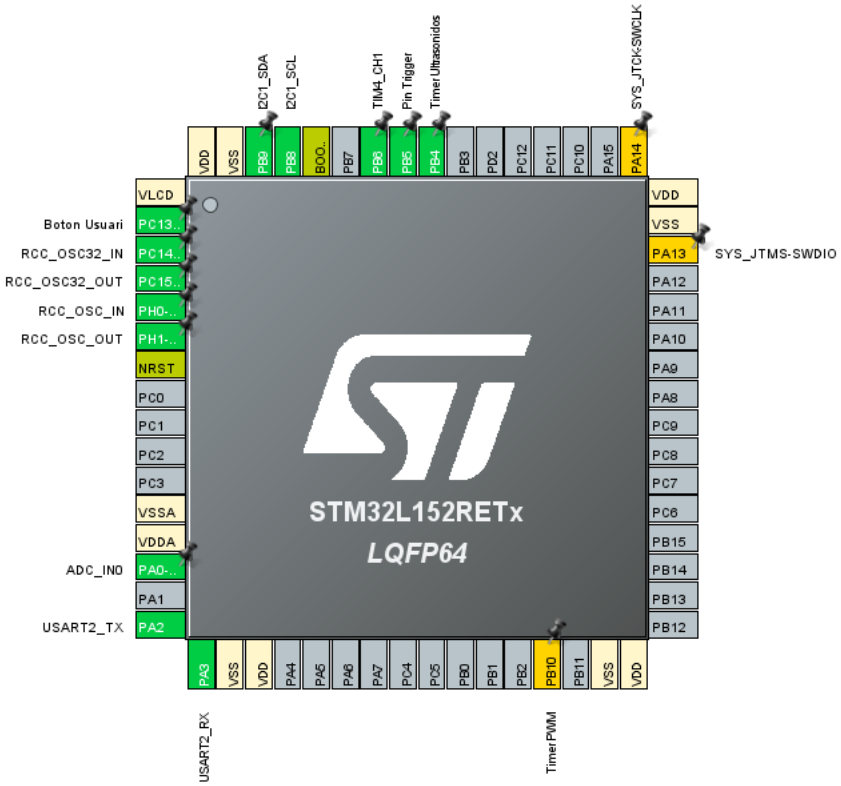

*Anexo 4 – Funcionamiento del los pulsos del sensor utrasonidos*

## Anexo 5



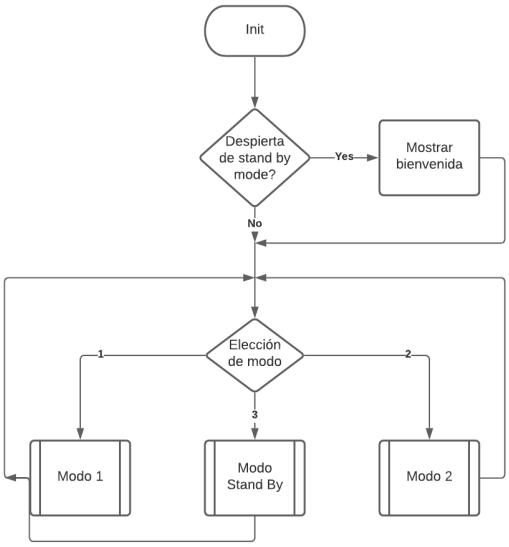

*Anexo 5 – Circuito con el pontenciómetro*

## Anexo 6




*Anexo 6 – Organización de pines en el microprocesadores*

## Anexo 7




*Anexo 7 – Diagrama de flujo principal*

## Anexo 8

```
//**Funcion para printf

        #ifdef __GNUC__
        /* With GCC, small printf (option LD Linker->Libraries->Small printf
         set to 'Yes') calls __io_putchar() */
        #define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
        #define GETCHAR_PROTOTYPE int __io_getchar(void)
        #else
        #define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
        #define GETCHAR_PROTOTYPE int fgetc(FILE *f)
        #endif /* __GNUC__ */

        PUTCHAR_PROTOTYPE {
            /* Place your implementation of fputc here */
            /* e.g. write a character to the EVAL_COM1 and Loop until the end of transmission */
            HAL_UART_Transmit(&huart2, (uint8_t*) &ch, 1, 0xFFFF);
            return ch;
        }
        GETCHAR_PROTOTYPE {
            /* Place your implementation of fgetc here */
            /* e.g. write a character to the EVAL_COM1 and Loop until the end of transmission */
            char ch;
            HAL_UART_Receive(&huart2, (uint8_t*) &ch, 1, 0xFFFF);
            return ch;
        }
```

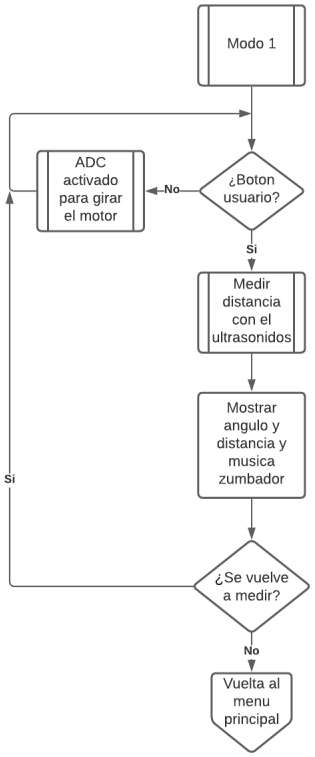
*Anexo 8 – Macro para enviar datos a la terminal*

## Anexo 9

```
128 //**Funcion para leer de la terminal
129
130         int readMsg(void)
131         {
132
133          for( len =0;len<256;len++)
134          {
135             while(!(USART2->SR & USART_SR_RXNE));
136            buff[len]= USART2->DR;
137             if (buff[len]==10 || buff[len]==13) break; //Condicion si se preciona enter
138          }
139          return len; //Devuelve el string de caracteres sin incluir el enter
140         }
141
```

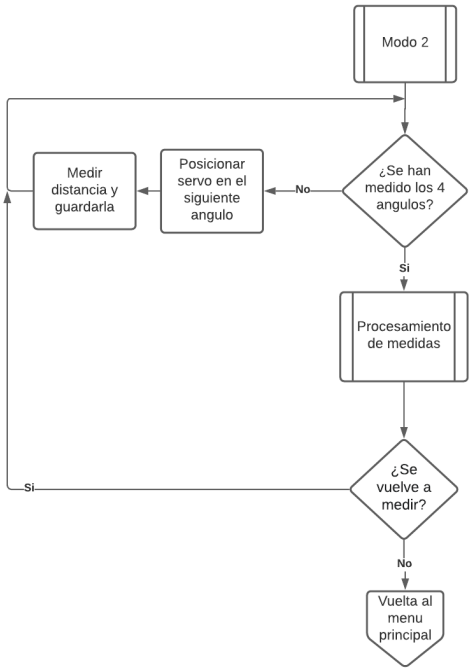
*Anexo 9 – Función para leer datos introducidos por la terminal*

## Anexo 10




*Anexo 10 – Diagrama de flujo del Modo 1*

## Anexo 11



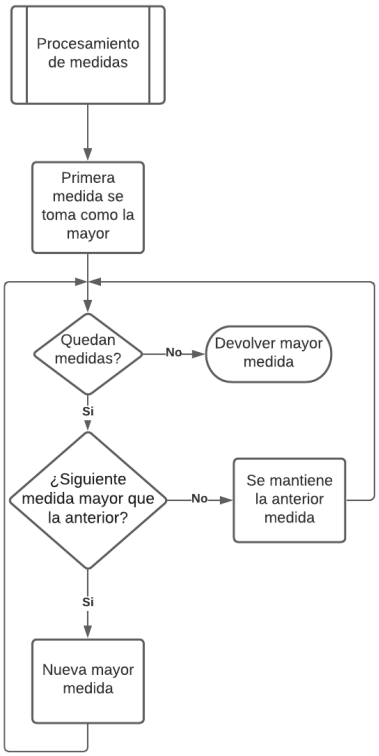

*Anexo 11 – Digrama de flujo del modo 2*

## Anexo 12

```
392                 for(int i=0;i<4;i++){
393                     //Posicionar el motor
394                     TIM2->CCR3=CCR_motor[i];
395 
396                     HAL_Delay(1500);
397                     //Enviar señal al trigger
398                     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);
399                     delay(3);
400                     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
401                     delay(10);
402                     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);
403                     HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_1);
404                     HAL_Delay(500);
405                     //Guardar la medida obtenida
406                     if(time_captured){
407                     //Calcular distancia y guardarlax
408                         distance = (time_elapsed)/58;
409                         distances[i]= distance;
410                         time_captured = 0;
411 
412                     }
413                 }
414                 done_measuring = 1;
415             }
```

*Anexo 12 – Código del for loop del Modo 2*

## Anexo 13




*Anexo 13 – Diagrama de flujo del analisis de medidas del Modo 2*

## Anexo 14

```
439
440             largest = distances[0];
441
442             //Buscar la posicion con mas distancia medida por el ultrasonidos
443             for(int i=0;i<4;i++){
444                 if(distances[i]>largest){
445                     largest = distances[i];
446                     largest_position = i;
447                 }
448             }
```

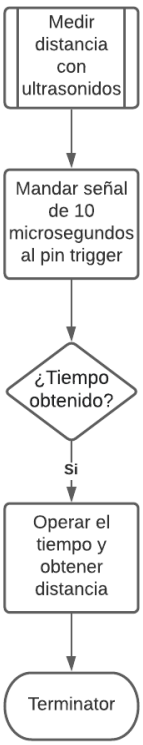
*Anexo 14 – Código del analisis de medidas del Modo 2*

## Anexo 15

```
233 //**Codigo para el standby mode
234
235         void gotoSleep(void){
236
237               //Enable the PWR Control Clock
238               RCC->APB1ENR|= (RCC_APB1ENR_PWREN);
239
240               //Set SLEEDEEP bit of Cortex System Control Register
241               SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;
242
243               //Select Standby mode
244               PWR->CR |= PWR_CR_PDDS;
245
246               //Clear wake up flag
247               PWR->CR |= PWR_CR_CWUF;
248
249               //enable wakeup pin
250               PWR->CR |= (PWR_CSR_EWUP2);
251
252               //Request wait for interrupt
253               __WFI();
254
255         }
```

*Anexo 15 – Código para entrar en Modo Stand By*

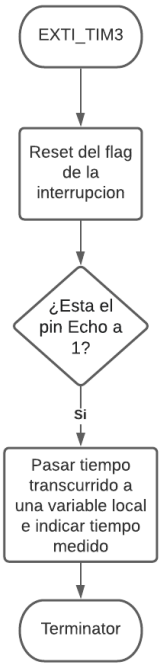
## Anexo 16




*Anexo 16 – Diagrama de flujo de medidas con el sensor de ultrasonidos*

## Anexo 17

```
208 void TIM3_IRQHandler(void)
209 {
210   /* USER CODE BEGIN TIM3_IRQn 0 */
211     if(TIM3 -> SR & TIM_SR_CC1IF){
212
213         TIM3 -> SR &= ~TIM_SR_CC1IF;
214
215         if(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_4)){
216             TIM3->CNT=0;
217         }else {
218
219             HAL_TIM_IC_Stop_IT(&htim3, TIM_CHANNEL_1);
220             time_elapsed = HAL_TIM_ReadCapturedValue(&htim3, TIM_CHANNEL_1);
221             time_captured = 1;
222         }
223     }
224
225   /* USER CODE END TIM3_IRQn 0 */
226   /* USER CODE BEGIN TIM3_IRQn 1 */
227
228   /* USER CODE END TIM3_IRQn 1 */
229 }
```




*Anexo 17 – Código y diagrama de flujo de la interrupción del timer 3*

## Anexo 18

```
//**Funcion de notas musicales

        void Correcto_zelda(void){
              TIM4->PSC = 319-1;
              htim4.Instance->CCR1 = 50;
              HAL_Delay(200);
              TIM4->PSC = 338-1;
              htim4.Instance->CCR1 = 50;
              HAL_Delay(200);
              TIM4->PSC = 402-1;
              htim4.Instance->CCR1 = 50;
              HAL_Delay(200);
              TIM4->PSC = 801-1;
              htim4.Instance->CCR1 = 50;
              HAL_Delay(200);
              TIM4->PSC = 603-1;
              htim4.Instance->CCR1 = 50;
              HAL_Delay(200);
              TIM4->PSC = 380-1;
              htim4.Instance->CCR1 = 50;
              HAL_Delay(200);
              TIM4->PSC = 301-1;
              htim4.Instance->CCR1 = 50;
              HAL_Delay(200);
              TIM4->PSC = 239-1;
              htim4.Instance->CCR1 = 50;
              HAL_Delay(400);
              htim4.Instance->CCR1 = 0;
        }
```

*Anexo 18 – Código para cancion del Modo 2*

## Anexo 19

```
//Initialize MPU6050 module and I2C
 MPU6050_Init(&hi2c1);

 //MPU6050 Config
 myMpuConfig.Accel_Full_Scale = AFS_SEL_2g;
 myMpuConfig.ClockSource = Internal_8MHz;
 myMpuConfig.CONFIG_DLPF = DLPF_184A_188G_Hz;
 myMpuConfig.Gyro_Full_Scale = FS_SEL_500;
 myMpuConfig.Sleep_Mode_Bit = 1; // 1 = sleep mode, 0= normal mode
 MPU6050_Config(&myMpuConfig);
```

*Anexo 19 – Codigo de configuración del MPU6050*

## Anexo 20

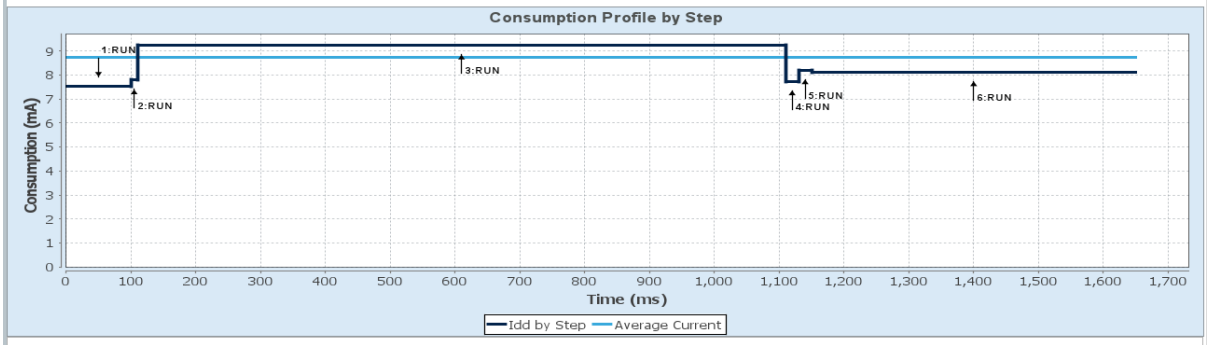
User Sequence Table

| Step | Mode | Vdd | Range/Scale | Memory | CPU/Bus Freq | Clock Config | Peripherals | Step Current | Duration |
|---|---|---|---|---|---|---|---|---|---|
| 1 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | | 7.55 mA | 100 ms |
| 2 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | USART2 | 7.79 mA | 10 ms |
| 3 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | ADC TIM2 | 9.24 mA | 1 s |
| 4 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | GPIOC | 7.75 mA | 20 ms |
| 5 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | GPIOB TIM3 TIM7 | 8.21 mA | 20 ms |
| 6 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | TIM4 USART2 | 8.13 mA | 500 ms |

Sequence Information Notes

Display Selection

Select your Preferred Display: Plot: All Steps




**Sequence Time / Ta Max** 1.65 s / 103.73 °C
**Battery Life Estimation** 16 days, 3 hours
**Average Consumption** 8.76 mA
**Average DMIPS** 33 DMIPS

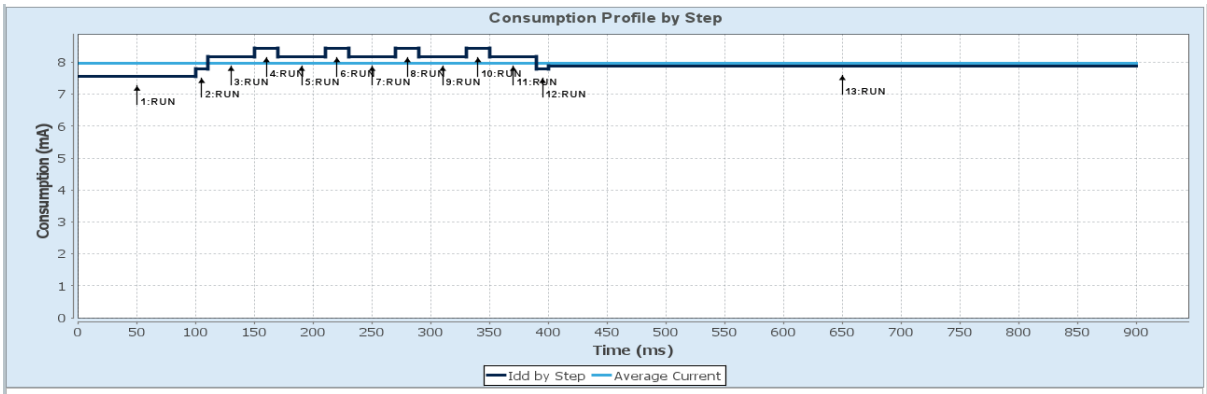
*Anexo 20 – Estudio del consumo energético del Modo 1*

## Anexo 21

User Sequence Table

| Step | Mode | Vdd | Range/Scale | Memory | CPU/Bus Freq | Clock Config | Peripherals | Step Current | Duration |
|---|---|---|---|---|---|---|---|---|---|
| 1 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | | 7.55 mA | 100 ms |
| 2 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | USART2 | 7.79 mA | 10 ms |
| 3 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | I2C1 TIM2 | 8.17 mA | 40 ms |
| 4 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | GPIOA I2C1 TIM3 ... | 8.44 mA | 20 ms |
| 5 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | I2C1 TIM2 | 8.17 mA | 40 ms |
| 6 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | GPIOA I2C1 TIM3 ... | 8.44 mA | 20 ms |
| 7 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | I2C1 TIM2 | 8.17 mA | 40 ms |
| 8 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | GPIOA I2C1 TIM3 ... | 8.44 mA | 20 ms |
| 9 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | I2C1 TIM2 | 8.17 mA | 40 ms |
| 10 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | GPIOA I2C1 TIM3 ... | 8.44 mA | 20 ms |
| 11 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | I2C1 TIM2 | 8.17 mA | 40 ms |
| 12 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | USART2 | 7.79 mA | 10 ms |
| 13 | RUN | 3.0 | Range1-High | RAM | 32 MHz | HSEBYP PLL | TIM4 | 7.88 mA | 500 ms |

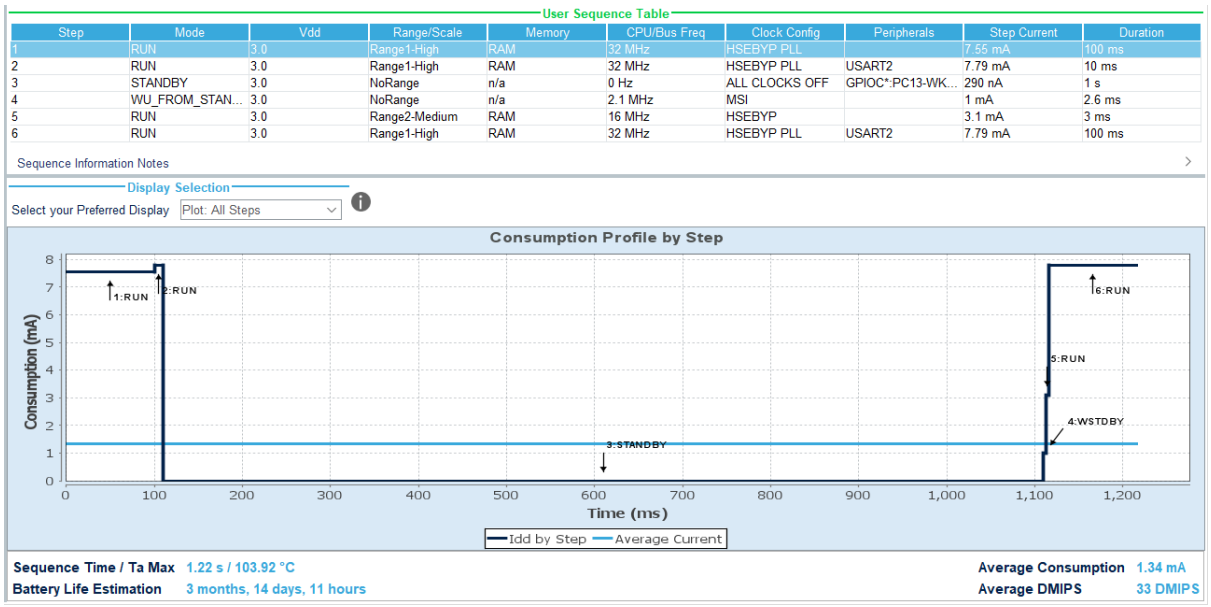
Sequence Information Notes




**Sequence Time / Ta Max** 900 ms / 103.83 °C
**Battery Life Estimation** 17 days, 19 hours
**Average Consumption** 7.96 mA
**Average DMIPS** 33 DMIPS

*Anexo 21 – Estudio del consume energético del Modo 2*

## Anexo 22




*Anexo 22 – Estudio del consume energético del Modo Stand By*

## 1.9 Bibliografía


- Controllers Tech. *How to Interface Buzzer with STM32 || PWM || HAL || CubeMx*. *Youtube*, 18 Dec. 2018, www.youtube.com/watch?v=AY8t7lsiG7I&ab_channel=ControllersTech.

- Eddie Amaya. *STM32 Standby / Sleep Mode Tutorial*. *Youtube*, 9 Sept. 2018, www.youtube.com/watch?v=O82rj9qxkgs&t=1303s&ab_channel=EddieAmaya.

- Mutex Embedded. *STM32F4 Discovery Board - Keil 5 IDE with CubeMX: Tutorial 35 - MPU6050 IMU Module*. *Youtube*, Mutex Embedded, 20 Apr. 2019, youtube.com/watch?v=-tvaeeJMW1w&t=1s&ab_channel=MutexEmbedded-Education.