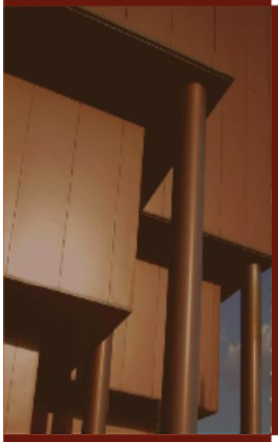
Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

Guía didáctica del diseño y creación de una espada laser

Autor: Borja Gordillo Ramajo

Tutor: Ignacio Alvarado Aldea



Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Guía didáctica del diseño y creación de una espada laser

Autor:
Borja Gordillo Ramajo

Tutor:
Ignacio Alvarado Aldea

Departamento de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Trabajo Fin de Máster: Guía didáctica del diseño y creación de una espada laser

Autor: Borja Gordillo Ramajo

Tutor: Ignacio Alvarado Aldea

El tribunal nombrado para juzgar el Trabajo Fin de Grado arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

Resumen

El trabajo fin de grado, trata de la creación de una guía didáctica, dividida en varias prácticas, para iniciarse en el mundo de Arduino, de la programación y de la electrónica a aquellas personas que no tienen los conocimientos básicos necesarios, ya que tratamos temas simples como el funcionamiento de un led o como trabajar con un botón, hasta casos más complejos, como las interrupciones o el uso de componentes externos compatibles con Arduino. Se irán explicando ejercicios, todos entrelazados entre sí, que crecerán en complejidad para acabar teniendo un conocimiento más amplio sobre todos los temas abordados y así ser capaz, como objetivo final, de crear una espada laser de forma autónoma y con la capacidad de poder hacer las mejoras y cambios que cada usuario desee. Por último, se expondrán posibles mejoras que se pueden hacer al funcionamiento que nosotros hemos desarrollado.

Índice

Capítulo 1. Introducción

1.1. Introducción

Arduino, es un proyecto libre que apareció hace unos años. Está basado en openhardware, eso quiere decir que sus especificaciones y diagramas esquemáticos son públicos, esto hace que lo estudiantes tengo un acceso fácil a él.

Ha diferencia de otros microcontroladores (PICs), Arduino no se programa a bajo nivel con ensamblador, sino que se usa un lenguaje mucho más conocido y comprensible para los usuarios, como es C/C++, y se conecta a través de un puerto USB para ser programada. Esto hace, que una persona que no sepa nada de PICs, puede programar un Arduino en poco tiempo.

La primera versión, UNO, contaba con 14 pines de E/S digitales (2 de ellos para conexión serie). Estos pines los usaremos para la mayoría de los sensores, botones, relés, etc… Solo tiene dos estados, encendido (HIGH) y apagado (LOW), También tiene 6 pines analógicos, con 1024 niveles de tensión. Estos nos sirven para leer sensores que devuelven rangos de tensiones dependiendo de su estado.

Con esta estructura que hemos resumido, se pueden hacer proyectos básicos de programación. Pero, para proyectos más elaborados, se necesita más potencia, por eso fueron sacando más versiones mejoradas.

1.2. Objeto

El fin del presente proyecto fin de grado es el de crear una guía didáctica, dividida en varias partes, para obtener, desde los conocimientos más básicos de Arduino, como puede ser encender un led, hasta saber trabajar con interrupciones provocadas por agentes externos. También se pretende exponer conocimientos básicos de electrónica, como el funcionamiento de un botón o la medición de consumo de unos leds, y automatización, ya que trabajaremos con diferentes estados para el correcto funcionamiento de nuestro proyecto.

Capítulo 2. Desarrollo del proyecto.

En este capítulo iremos desarrollando los pequeños proyectos que se van necesitando para llegar a nuestro proyecto final, que es poder construir una espada laser casera.

2.1. Botón, leds e interrupciones

En esta primera parte, explicaremos el funcionamiento siempre de un botón, como trabajar con la tira de leds WS2812B y aprenderemos el concepto y el uso de las interrupciones de Arduino

2.1.1. Funcionamiento de un botón

En primer lugar, veremos el funcionamiento de un botón. Normalmente, para poder conectar un pulsador al Arduino, necesitaríamos añadir una resistencia como se ve en la figura 1, de forma que cuando el interruptor se cierre, la tensión en la entrada digital del Arduino será 0V, y cuando se abra, gracias a la resistencia añadida, se pondrá a 5V. A este tipo de resistencias se le llaman resistencias PULL UP. Arduino posee este tipo de resistencias de forma interna y podemos conectarlas mediante software (programación). Para ello, declararemos el pin al que conectemos dicha entrada, en vez de:

$$pinMode(3, INPUT)$$

Lo declararemos como:

$$pinMode(3, INPUT\_PULLUP)$$

Y Arduino conectará una resistencia de 10KΩ entre dicho pin y 5V. de esta forma podemos conectar directamente el cable desde el botón al pin del Arduino.

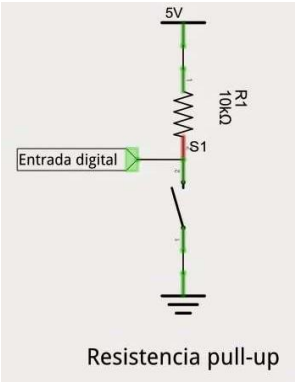


Figura 1: Resistencia PULL UP

Pero este montaje tiene un problema, el efecto rebote o bouncing, que hace que la tensión fluctúe entre 0V y 5V, pudiendo encontrarse falsos positivos.

Para corregir este problema, usaremos el método de debouncing por hardwere. Este método, consiste en colocar un condensador en paralelo con el pulsador (ver figura 2). El condensador tardará un tiempo en cargarse y una vez que esté cargado, la señal de entrada estará a 5V.

Para calcular el tiempo que tarda el condensador en cargarse, usaremos la siguiente formula:

$$\tau = R \times C$$

Esta aproximación es buena, ya que, para Arduino, un pin está HIGH cuando sobre pasa el 60% de la tensión de funcionamiento, en nuestro caso serían 3V porque trabajamos con 5V, y τ es la constante que define el tiempo que se tardará en cargar un condensador al 63,2% de su capacidad. Por lo tanto, teniendo en cuenta la resistencia interna de Arduino, podemos usar un condensador de 5µF.

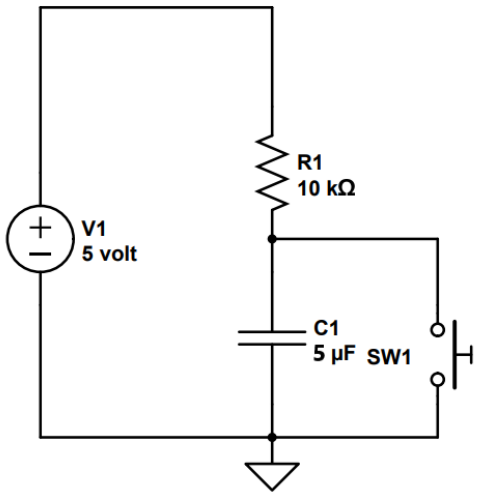


Figura 2: Esquema Debouncing

A continuación, mostraremos como sería el montaje entero en la figura 3, donde conectaremos el led con una resistencia y el botón con la resistencia interna.

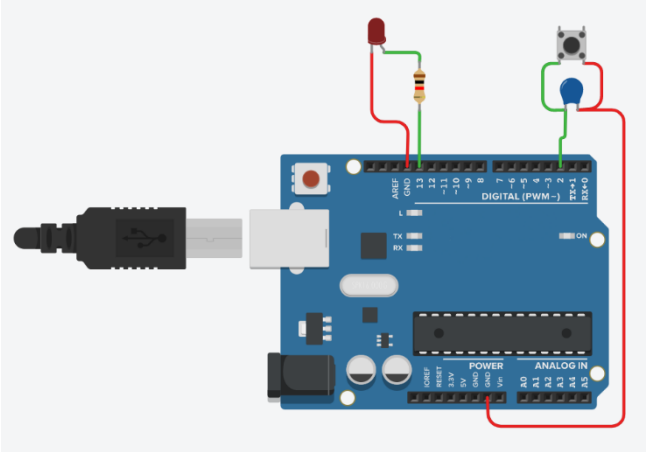


Figura 3: Esquema montaje Botón-Led

Para obtener el código necesario para este código, buscaremos en la siguiente ruta en nuestro programa de Arduino: *Archivo* > *Ejemplos* > *02.Digital* > *Button*

```
1   // constants won't change. They're used here to set pin numbers:
2   const int buttonPin = 2;     // the number of the pushbutton pin
3   const int ledPin =  13;      // the number of the LED pin
4
5   // variables will change:
6   int buttonState = 0;         // variable for reading the pushbutton
7   status
8
9   void setup() {
10    // initialize the LED pin as an output:
11    pinMode(ledPin, OUTPUT);
12    // initialize the pushbutton pin as an input:
13    pinMode(buttonPin, INPUT);
14  }
```

```
1
2   void loop() {
3     // read the state of the pushbutton value:
4     buttonState = digitalRead(buttonPin);
5
6     // check if the pushbutton is pressed. If it is, the buttonState is
7   HIGH:
8     if (buttonState == HIGH) {
9       // turn LED on:
10      digitalWrite(ledPin, HIGH);
11    } else {
12      // turn LED off:
13      digitalWrite(ledPin, LOW);
14    }
15  }
```

El código es muy simple:

- En las líneas 2 y 3 declaramos las constantes para conectar el botón y el led.
- En la línea 6 declaramos el estado del botón, que en este caso es 0 (0V).
- En la línea 9 empieza una de las funciones principales, que es la función *setup()*, que solo se ejecutará una vez. Dentro de ella, declaramos el pin del led como salida (*OUTPUT*) y el del botón como entrada, pero tenemos que modificar el tipo, ya que hemos dicho que lo conectaremos a una resistencia interna, por lo que tendremos que cambiar de *INPUT* a *INPUT_PULLUP*.
- En la línea 16 tenemos la segunda función principal, la función loop(), que se ejecutará en bucle. En ella, en la línea 18, leemos el estado del botón y en la línea 22 comprobamos si está pulsado (HIGH). Si esta pulsado, encendemos el led escribiendo HIGH (línea 24), en caso contrario, apagamos el led escribiendo LOW (línea 27).

2.1.2. Leds WS2812B.

Los WS2812B son leds que disponen de lógica integrada, por lo que es posible variar el color de cada led de forma individual.

Están basados en el led 5050, llamado así porque tiene un tamaño de 5.0 x 5.0 mm. Es un led de bajo consumo y alto brillo, que incorpora en un único encapsulado los 3 colores primarios (RGB).

La genial novedad del WS2812B (y el resto de familia) es añadir un integrador dentro de cada led, que permite acceder a cada píxel de forma individual. Por este motivo, este tipo de leds se denominan *individual addressable*.

Para su correcto funcionamiento, cada led dispone de un integrador que almacena 3 bytes (24 bits), que corresponden con los 3 colores del RGB. Cada píxel puede tener 256 niveles en cada color, lo que supone un total de 16.777.216 posibles colores.

Esta genial idea permite hacer configuraciones de múltiples leds, en los que únicamente tenemos que comunicarnos con el primero de ellos y cada led actúa de transmisor de la secuencia a los leds posteriores. Además, permite que podamos encadenar o dividir tiras de leds y cualquier fragmento seguirá funcionando porque todos los leds tienen exactamente el mismo comportamiento.

Cada vez que un punto transmite al siguiente una señar, realiza una reconstrucción de forma que la distorsión y el ruido no se acumulan, esto permite alimentar tiras de más de 5m sin necesidad de dispositivos adicionales

La frecuencia de funcionamiento es superior a 400Hz/s. Esto permite que se puedan animar más de 1024 puntos a una tasa de refresco de 30fps.

El esquema de montaje de estos leds es muy simple:

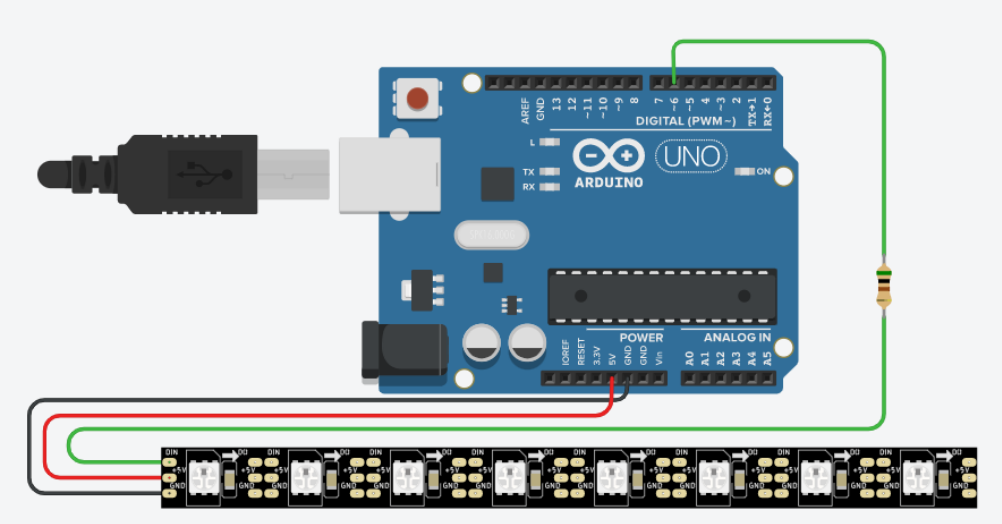


Figura 4: Esquema montaje TiraLeds

Es muy importante colocar una resistencia (300Ω - 500Ω) entre el pin de dato y el pin de entrada de la tira de leds para ayudar a prevenir picos de voltajes que podrían dañar los leds.

En la parte de programación, lo primero que tendremos que hacer es instalar la librería necesaria para trabajar con esta tira de led. Para ello, desde nuestro IDE haremos los siguientes pasos:

1. Seleccionar: *Programa* > *Incluir Librería* > *Administrar Bibliotecas…*
2. Se nos abrirá una ventana. En el cuadro donde pone *Filtre su búsqueda*, escriba el nombre de la librería, búsquela e instálela.

A continuación, abra el ejemplo que se encuentra en la siguiente ruta*: Archivo > Ejemplos > Adafruit NeoPixel > simple*.

```
#include <Adafruit_NeoPixel.h>
#ifdef __AVR__
#include <avr/power.h> // Required for 16 MHz Adafruit Trinket
#endif

// Which pin on the Arduino is connected to the NeoPixels?
#define PIN        6 // On Trinket or Gemma, suggest changing this to
1

// How many NeoPixels are attached to the Arduino?
#define NUMPIXELS 16 // Popular NeoPixel ring size

// When setting up the NeoPixel library, we tell it how many pixels,
// and which pin to use to send signals. Note that for older NeoPixel
// strips you might need to change the third parameter -- see the
// strandtest example for more information on possible values.
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);

#define DELAYVAL 500 // Time (in milliseconds) to pause between pixels
void setup() {
  // These lines are specifically to support the Adafruit Trinket 5V
16 MHz.
  // Any other board, you can remove this part (but no harm leaving
it):
#if defined(__AVR_ATtiny85__) && (F_CPU == 16000000)
  clock_prescale_set(clock_div_1);
#endif
  // END of Trinket-specific code.

  pixels.begin(); // INITIALIZE NeoPixel strip object (REQUIRED)
}

void loop() {
  pixels.clear(); // Set all pixel colors to 'off'

  // The first NeoPixel in a strand is #0, second is 1, all the way up
  // to the count of pixels minus one.
  for(int i=0; i<NUMPIXELS; i++) { // For each pixel...

    // pixels.Color() takes RGB values, from 0,0,0 up to 255,255,255
    // Here we're using a moderately bright green color:
    pixels.setPixelColor(i, pixels.Color(0, 150, 0));

    pixels.show();   // Send the updated pixel colors to the hardware.

    delay(DELAYVAL); // Pause before next pass through loop
  }

}
```

Para poder ejecutar nuestro programa, tenemos que hacer las siguientes modificaciones:

1- Tenemos que modificar NUMPIXELS por el número de leds total que tiene nuestra tira de leds.
2- Tenemos que modificar el valor de PIN con el valor del pin al que hemos conectado el pin de datos de la tira de leds.

Una vez configurado el Arduino para trabajar con nuestra tira de leds, vamos a entender cómo funciona el código anterior.

1- En la línea 1, vemos como se incluye una librería en el código de Arduino.
2- En las líneas 7 y 11 tenemos otra forma de definir constantes.
3- En la línea 17, declaramos la tira de leds (*pixels*). Se le pasan las siguientes variables: número de leds, pin al que conectamos la tira de leds y configuración con la que vamos a trabajar (dejamos la que viene por defecto).
4- En la función *setup()*, en la línea 30, inicializamos la variable *pixels*.
5- En la línea 34, apagamos todos los leds.
6- En la línea 38, tenemos un *for*, que por cada led hace las siguientes instrucciones:
    - En la línea 42 declara el color que quiere para el led número 'i'.
    - En la línea 44 enciende dicho pin.
    - En la línea 46 ponemos un *delay()* para que veamos cómo se encienden progresivamente.

Para declarar el color de un led, tenemos dos formas:

1- Como hemos visto en el código anterior: *pixels.setPixelColor(i, pixels.Color(R,G,B))*. Las siglas RGB corresponden a los colores primarios (R = red, G = green y B = blue).
2- La otra forma, es declarar directamente el color: *pixels.setPixelColor(i,R,G,B)*.

2.1.3. Interrupciones

Una interrupción es una parada temporal de la ejecución del programa principal, para pasar a ejecutar una subrutina (de servicio de interrupción), la cual, por lo general, no forma parte del programa, sino que realiza otra tarea. Una vez finalizada dicha subrutina, se reanuda la ejecución del programa exactamente donde se dejó.

Para entender la utilidad y necesidad de las interrupciones, supongamos que tenemos Arduino conectado a un sensor, por ejemplo, encoder óptico que cuenta las revoluciones de un motor, un detector que emite una alarma de nivel de agua en un depósito, o un simple pulsador de parada.

Si queremos detectar un cambio de estado en esta entrada, un posible método es consultar las entradas digitales repetidamente, con un intervalo de tiempo entre consultar. Este mecanismo se denomina 'poll', y tiene 3 claras desventajas:

1- Supone un continuo consumo de procesador y de energía, al tener que preguntar continuamente por el estado de la entrada.
2- Si la acción necesita ser atendida inmediatamente, por ejemplo, en una alerta de colisión, esperar hasta el punto de programa donde se realiza la consulta puede ser inaceptable.
3- Si el pulso es muy corto, o si el procesador está ocupado haciendo otra tarea mientras se produce, es posible que nos saltemos el disparo y nunca lleguemos a verlo.

Para resolver este tipo de problemas, los microprocesadores incorporan el concepto de interrupción, que es un mecanismo que permite asociar una función a la ocurrencia de un determinado evento.

Cuando ocurre el evento el procesador 'sale' inmediatamente del flujo normal del programa y ejecuta la función de interrupción asociada ignorando por completo cualquier otra tarea (por esto se llama interrupción). Al finalizar la función, el procesador vuelve al flujo principal, en el mismo punto donde había sido interrumpido.

2.1.3.1. Rutina de interrupción

Las rutinas de interrupción tienen algunas limitaciones:

1- Dos rutinas de interrupción no pueden ejecutarse de forma simultánea. En caso de dispararse una segunda interrupción mientras se ejecuta una, se ejecutará la que sea de más prioridad primero. En caso de que la segunda sea de más prioridad que la primera, ésta se interrumpe (ver sección 2.3.3) y continuará cuando acabe la segunda.
2- Las rutinas de interrupción han de ser lo más breves posibles.
3- No pueden devolver parámetros, ni tampoco recibirlos. Pero si pueden trabajar con variables globales, aunque estas, al declararse, han de ser de tipo 'volatile'. Estrictamente hablando, no es un tipo de variable, si no una directiva para el compilador. Es una manera de tratar la variable que previene errores (véase *https://www.arduino.cc/en/reference/volatile*).
4- Hay determinadas funciones que no se pueden usar dentro de esta rutina, ya que usan otras interrupciones. Es el caso de funciones como: *delay(), millis(), micros(), Serial.print(), Serial.println(), Serial.write() y Serial.read()*.
5- Debe evitarse el uso de funciones analogRead() y analogWrite(), ya que tienen un impacto serio sobre la velocidad, y como hemos dicho anteriormente, la función debe ser corta y rápida.

2.1.3.2. Interrupciones externas o asociadas al cambio de un pin

Como ejemplo de aplicación, vamos a medir el tiempo que se mantiene un botón pulsado.

Usando el procedimiento descrito (polling), se puede resolver el problema perfectamente, pero utilizando interrupciones, consumiremos meno tiempo del micro.

El modo de atender a esto cambios mediante una interrupción es sencillo, pero hay que tener algunas cosas en cuenta:

1- En Arduino Uno (y en el nano) sólo hay dos pines de interrupción externa, el pin 2 (correspondiente a la interrupción externa cero, INT0) y el pin 3 (correspondiente a la interrupción externa uno, INT1). En otras versiones de Arduino, hay un número diferente de pines que se pueden utilizar como interrupciones externas, en el caso del Mega hay 6 pines. También hay librerías que te dejan usar conjuntos de pines para activar determinadas interrupciones.
2- Estas interrupciones pueden ajustarse para disparar por nivel o por flancos ascendentes o/y descendentes de la señal:
   - LOW: se activa cuando la tensión del pin es LOW (0V).
   - HIGH: se activa cuando la tensión del pin es HIGH (5V).
   - CHANGE: se activa cuando la tensión del pin cambia de HIGH → LOW o de LOW → HIGH.
   - RISING: se activa cuando la tensión del pin cambia de LOW → HIGH.
   - FALLING: se activa cuando la tensión del pin cambia de HIGH → LOW.

Para configurar las interrupciones externas, se usa la función:

attachInterrupt(< INT >, < nombre_func >, < forma de act>)

Esta función ha de utilizarse dentro de *setup()*. Donde:

1- En < INT > habrá que poner el número de interrupción (INT0 si conectamos al pin 2 o INT1 si conectamos al pin 3).
2- En < nombre_func > pondremos el nombre de la función que queremos que se ejecute cuando salte la interrupción.
3- En < forma de act > pondremos LOW, HIGH, CHANGE, RISING o FALLING.

La función de la interrupción externa tiene casi las mismas restricciones que la función de interrupción temporal que vimos en la sección 2.1.3.1

2.1.3.3. Prioridad entre interrupciones

Existen muchas fuentes para generar interrupciones En el ATmega328p (microprocesador del Arduino Uno). En el caso de que existan distintas interrupciones simultaneas, el microprocesador atenderá primero, aquellas que tengan mayor prioridad. En la figura 5, podemos ver la tabla con las prioridades de las interrupciones. La s interrupciones de mayor prioridad son las que están más alto en la tabla.

VectorNo.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

Figura 5: Tabla de interrupciones

2.1.4. Problema interrupciones 1

En este primer problema, vamos a plantear hacer un programa que, mediante interrupciones, nos imprima por el puerto serie el tiempo que ha estado pulsado el botón. Definir un umbral de tiempo (Tm = 500 ms), de modo que distingamos entre pulsación corta y larga. Imprimir por el puerto serie si ha sido larga o corta.

Daremos algunas indicaciones para facilitar el primer problema. La interrupción que usaremos será CHANGE y dentro de la función de interrupción leeremos el estado del pin. Si el estado es HIGH, significa que acabamos de pulsar, si, por el contrario, el estado es LOW, significa que acabamos del soltar (hay que tener en cuenta, que la función de interrupción, solo se ejecuta cuando cambia el estado del pin). Cuando soltemos, además podremos calcular el tiempo pulsado y decidir que se hace en consecuencia.

2.1.5. Problema interrupciones 2

Para este caso, vamos a complicar el uso del botón, de forma qué:

- Si pulso el botón más de un segundo, sin necesidad de soltar, habrá una función que va a devolver un 1.
- Si la pulsación es menor de un segundo, no devolvemos nada hasta que contemos las pulsaciones que se dan durante 3 segundos, y devolveremos ese número.

La idea es la siguiente:

- Si se devuelve un 1, encendemos la espada si estaba apagada y la apagamos si estaba encendida (sacamos por el puerto serie un mensaje que ponga, espada encendida o apagada).
- Si se devuelve un 3, la espada ha de cambiar de color (sacamos por el puerto serie un mensaje que ponga, Cambio de color).
- Si se devuelve un 5, la espada cambiara de sonido (sacamos por el puerto serie un mensaje que ponga, Cambio de sonido).

2.1.6. Problema espada laser 1

Los objetivos de este problema son 3:

1- Cuando pulsemos el botón más de TM = 500 milisegundos (sin esperar a soltar), si la espada está apagada, se encenderá, y si está encendida, se apagará. Llamaremos a una función que hará que los leds se vayan encendiendo de la base al extremo o apagando del extremo a la base.
   Téngase en cuenta, a la hora de programar estas funciones, la disposición de los leds dentro de la hoja de la espada. Es una única tira de leds doblada por la mirad en forma de U, por lo que los primeros leds en encenderse serán los colocados en los dos extremos, y los últimos, los colocados en el medio a la hora de simular el encendido de la espada.
2- Cuando se devuelva un 3, la espada, si esta encendida, cambia de entre una sección de colores que hayamos elegido previamente.
3- Cuando se devuelva un 5, la espada, si está encendida, cambia de sonido. Ahora mismo no se puede hacer, pero dejamos el programa preparado para cuando sepamos hacerlo.

2.2. Unidad inercial MPU-6050

La MPU-6050 es una unidad de medición inercial (IMU) de seis grados de libertad (6DOF) fabricado por Invense, que combina un acelerómetro de 3 ejes y un giroscopio de 3 ejes. La comunicación puede realizarse tanto por SPI como por el bus I2C, por lo que es sencillo obtener los datos medidos. La tensión de alimentación es de bajo voltaje, entre 2.4V a 3.6V. Frecuentemente, se encuentran integrado en módulos como el GY-521, que incorporan la electrónica necesaria para conectarla de forma sencilla a un Arduino. En la mayoría de los módulos, esto incluye un regulador de voltaje que permite alimentar directamente a 5V.

Dispone de conversores analógicos digitales (ADC) de 16bits. El rango del acelerómetro puede ser ajustado a ±2g, ±4g, ±8g y ±16g, el giroscopio a ±250, ±500, ±1000 y ±2000º/s.

Es un sensor que consume 3.5m!, con todos los sensores y el DMP activados. Dispone de un sensor de temperatura embebido, un reloj de alta precisión e interrupciones programables. También puede conectarse a otros dispositivos I2C como master.

Wl MPU-6050 incorpora un procesador interno (DMP Digital Motion Processor) que ejecuta complejos algoritmos de MotionFusion para combinar las mediciones de los sensores internos, evitando tener que realizar los filtros de forma exterior.

El MPU-6050 es uno de los IMUs más empleados por su gran calidad y precio. Será uno de los componentes que con mayor frecuencia incorporaremos a nuestros proyectos de electrónica y robótica.

2.2.1. Esquema de montaje

La conexión es sencilla, simplemente alimentamos el módulo desde el Arduino mediante GND y 5V, y conectamos el pin SDA y SCL de Arduino con los pines correspondientes del sensor (ver figura 6).

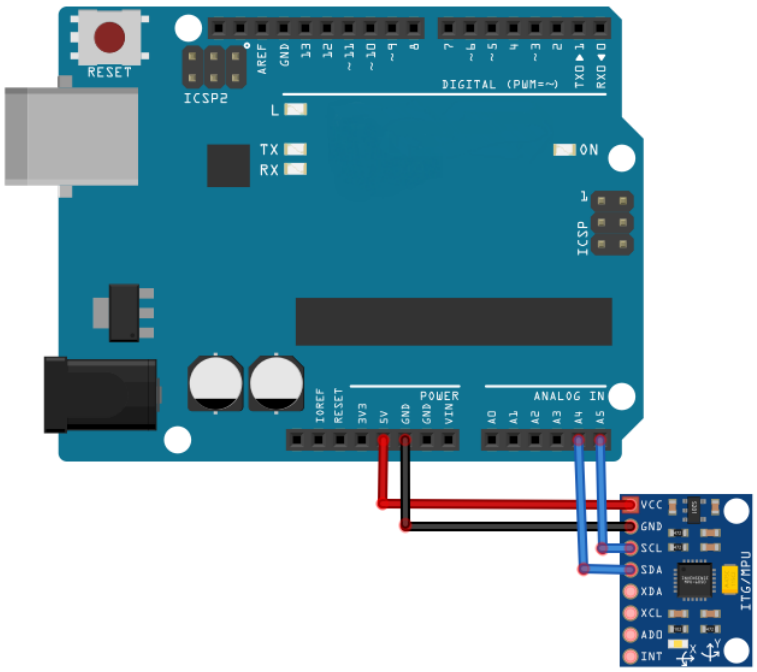


Figura 6: Esquema montaje MPU-6050

2.2.2. Representación de los valores

Una vez que hemos leído los valores que nos devuelve el MPU-6050, vamos a ver dos formas diferentes de mostrar estos valores: en bruto y escalados.

2.2.2.1. Valores en Bruto

Lo primero que debemos hacer, es instalar las librerías MPU6050.h e I2Cdev.h como vimos en el apartado 2.1.2.

Una vez tengamos las librerías instaladas, en el primer ejemplo, aprenderemos a leer los valores directamente proporcionados por el MPU-6050 (valores RAW) a través del bus I2C. Los valores RAW tienen un rango de medición entre -32768 y +32767.

```
1  #include "I2Cdev.h"
2  #include "MPU6050.h"
3  #include "Wire.h"
4
5  const int mpuAddress = 0x68;  //Puede ser 0x68 o 0x69
6  MPU6050 mpu(mpuAddress);
7
8  int ax, ay, az;  //Aceleraciones
9  int gx, gy, gz;  //velocidades angulares (no ángulos)
10
11 void printTab()
12 {
13    Serial.print(F("\t"));
14 }
15
16 void printRAW()
17 {
18    //Serial.print(F("a[x y z] g[x y z]:t"));
19    Serial.print(ax); printTab();
20    Serial.print(ay); printTab();
21    Serial.print(az); printTab();
22    Serial.print(gx); printTab();
23    Serial.print(gy); printTab();
24    Serial.println(gz);
25 }
26
27 void setup()
28 {
29    Serial.begin(9600);
30    Wire.begin();
31    mpu.initialize();
32    Serial.println(mpu.testConnection() ? F("IMU iniciado
33 correctamente") : F("Error al iniciar IMU"));
34 }
35
36 void loop()
37 {
38    // Leer las aceleraciones y velocidades angulares
39    mpu.getAcceleration(&ax, &ay, &az);
40    mpu.getRotation(&gx, &gy, &gz);
41
42    printRAW();
43
44    delay(100);
45 }
```

El código es muy simple:

- En la línea 5, declaramos la dirección del MPU-6050 y en la línea seis declaramos el MPU-6050.
- En la línea 11, creamos una función para tabular.
- En la línea 16, creamos una función que nos va pintando todos los valores, en bruto, de la velocidad y la aceleración.
- En el *setup()*, inicializamos la variable *mpu* e inicializamos la variable *Wire*. La variable Wire nos permite comunicarnos con otros dispositivos mediante I2C.
- En el *loop()*, lo que hacemos es leer las aceleraciones (*mpu.getAcceleration()*) y las velocidades angulares (*mpu.getRotation()*), y luego mostrarlas por el puesto Serial (*printRAW()*).

2.2.2.2. Valores escalados

Para obtener los valores escalados tendríamos que hacer dos pequeñas modificaciones en nuestro código:

- Añadir los dos factores de conversión:

```
// Factores de conversion
const float accScale = 2.0 * 9.81 / 32768.0;

const float gyroScale = 250.0 / 32768.0;
```

- Modificar la función *printRAW()* para que tenga en cuenta la escala:

```
// Mostrar medidas en Sistema Internacional
void printRAW()
{
   //Serial.print(F("a[x y z](m/s2) g[x y z](deg/s):t"));
   Serial.print(ax * accScale); printTab();
   Serial.print(ay * accScale); printTab();
   Serial.print(az * accScale); printTab();
   Serial.print(gx * gyroScale);  printTab();
   Serial.print(gy * gyroScale);  printTab();
   Serial.println(gz * gyroScale);

}
```

2.2.3. Problema cálculo umbrales

Una vez sabemos obtener los valores de los 6 ejes, necesitamos saber el valor absoluto de la velocidad angular y la aceleración.

Para ello, vamos a sacar, usando Serial Plotter (es una herramienta que hace gráficas con las variables que mandemos por el puerto serie) del IDE de Arduino, dos gráficas, una con la suma de los módulos de las tres velocidades y otra con la suma de los módulos de las tres aceleraciones. Hay que decidir 3 umbrales, uno para la velocidad y dos para la aceleración.

El umbral de la velocidad nos va a ayudar a distinguir si la espada se mueve o no.

Los umbrales de la aceleración nos van a ayudar a dos cosas:

1- El primero, que será más bajo, nos ayudará a decidir si el movimiento es lo suficientemente rápido como para hacer un sonido de movimiento.
2- El segundo, que será mas alto, nos va a ayudar a decidir si hay impacto o no.

Añadiremos a nuestro programa, el de la sección 2.2.2.2, en la función *setup()*, debajo de:

mpu.initializa();

La sentencia:

mpu.setFullScaleAccelRange(MPU6050_ACCEL_FS_16);

Que establece el máximo rango de medida para la aceleración. Habrá que actualizar pues la variable:

*const float accScale = 16.0*9.81/32769.0;*

También se recomienda cambiar el tipo de las variables *ax, ay, az, gx, gy y gz* de *int* a *int16*, porque es el tipo de dato que nos da la MPU6050 directamente. En realidad, es lo mismo, pero el número de bits del *int* depende de la máquina, mientras que el de *int16* siempre tiene 16 bits, independientemente de la máquina.

2.2.4. Problema movimiento o impacto

Una vez establecidos los umbrales, cuando se superen los umbrales establecidos para el movimiento, se devolverá un 1, y cuando se supere el umbral establecido para el impacto se devolverá un 2. En caso de que se superen ambos umbrales, devolveremos un 2, ya que un impacto siempre tendrá más relevancia.

Puede hacer que se encienda el led asociado al pin13 cuando la espada se mueva y que salga por el puerto serie un mensaje cuando detecte un impacto.

Se recomienda que se definan estas constantes con un #define al principio de su código, por si luego hay que retocar sus valores.

2.2.5. Problema espada laser 2

El siguiente paso sería añadir este módulo y su funcionalidad a nuestro proyecto. Es decir, complementar el código de la sección 2.1.6. con lo que hemos aprendido en esta sección y así poder cambiar los colores de la espada cuando esta sufre un impacto.

Puede ponerse un *delay()* entre el encendido y apagado de los leds cuando hay un impacto, para que se aprecie bien el cambio de color.

Por lo tanto, tenemos 3 posibles funcionalidades con respecto a los leds:

1- Los encendemos/apagamos cuando mantenemos el botón más de *Tm* milisegundos pulsados.
2- Cambiamos los colores de quito o movimiento, que son el mismo, e impacto, cuando pulsamos el botón 3 veces consecutivas.
3- Cambiamos el color de la espada al color de impacto durante un pequeño periodo de tiempo cuando detectamos un impacto.

2.3. Reproducción de sonidos desde una tarjeta SD

Para reproducir el sonido de nuestro proyecto usaremos un reproductor llamado DFPlayerMini. Las razones por las que hemos usado este módulo son su pequeño tamaño, la opción de poder usar un pequeño altavoz (3W o menos) y las diferentes posibilidades de reproducir sonidos, aunque nosotros solamente usaremos una SD para reproducir sonidos.

2.3.1. Reproductor DFPlayerMini

El reproductor DFPlayerMini es un reproductor que se puede usar con Arduino o como un reproductor de música independiente y tiene un bajo coste

Es un módulo muy completo:

- Reproduce formatos de ficheros MP3, WMA y WAV.
- El lector microSD es compatible con FAT16 y FAT32, con una capacidad máxima de 32GB.
- Soporta hasta 100 carpetas y puede acceder hasta 255 canciones.
- Proporciona velocidades de muestreo de 8, 11.025, 12, 16, 22.05, 24, 32, 44.1 y 48 KHz.
- Salida con DAC de 24 bits.
- Dispone de 30 niveles de volumen ajustable.
- Ecualizador de 6 niveles.
- Señal de ruido (SNR) de 85dB.

Su esquema de pines es el siguiente:

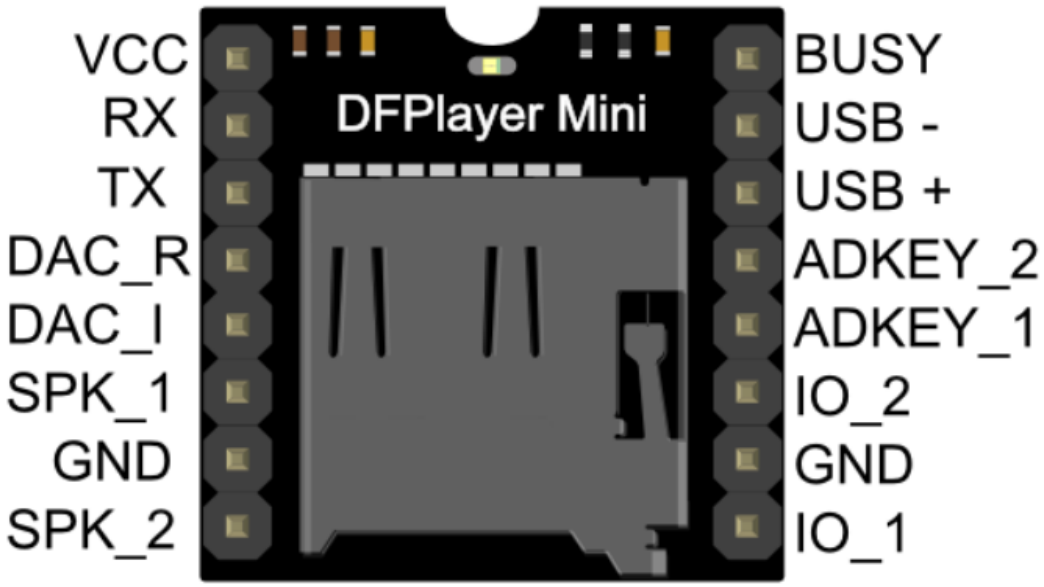


Figura 7: Esquema pines DFPlayerMini

A continuación, pondremos una tabla con una breve explicación de todos los pines, aunque explicaremos más detenidamente los que usaremos nosotros en nuestro proyecto.

Pin	Description	Note
VCC	Input Voltage	DC3.2~5.0V;Type: DC4.2V
RX	UART serial input	
TX	UART serial output	
DAC_R	Audio output right channel	Drive earphone and amplifier
DAC_L	Audio output left channel	Drive earphone and amplifier
SPK2	Speaker-	Drive speaker less than 3W
GND	Ground	Power GND
SPK1	Speaker+	Drive speaker less than 3W
IO1	Trigger port 1	Short press to play previous（long press to decrease volume）
GND	Ground	Power GND
IO2	Trigger port 2	Short press to play next（long press to increase volume）
ADKEY1	AD Port 1	Trigger play first segment
ADKEY2	AD Port 2	Trigger play fifth segment
USB+	USB+ DP	USB Port
USB-	USB- DM	USB Port
BUSY	Playing Status	Low means playing \High means no

Figura 8: Descripción pines

Los pines que nos interesan a nosotros son los siguientes:

- VCC y GND: para alimentar el componente (como máximo a 5V y 1A). Si usamos alimentación externa, siempre poner con GND común.
- SPK1 y SPK2: para conectar nuestro altavoz. Hay que recordar que debe ser menor de 3W, si no tendríamos que usar DAC_R y DAC_L para poder conectarnos a un altavoz con un conector Jack.
- RX y TX: son los pines para comunicarnos con Arduino.
- BUSY: es el pin que nos indica si hay algún sonido reproduciéndose.

Una cosa que debemos tener en cuenta es el nombre que le vamos a poner a los archivos de sonidos que tenemos guardados dentro de nuestra SD. Deben tener el siguiente formato, 4 dígitos numéricos y luego lo que nosotros queramos, pero solo sería obligatorio los 4 dígitos.

Para hacer referencia a uno en concreto, tenemos que escribir el número generado con los 4 dígitos, pero sin los ceros que queden a la izquierda, es decir, si guardamos una canción con el nombre 0020, nos referimos a ella como 20 (ejemplo: si tenemos guardado un sonido como 0002-ON, tenemos que llamarlo como 2).

2.3.2. Esquema de montaje

Las conexiones entre el DFPlayerMini y el Arduino serían las siguientes:

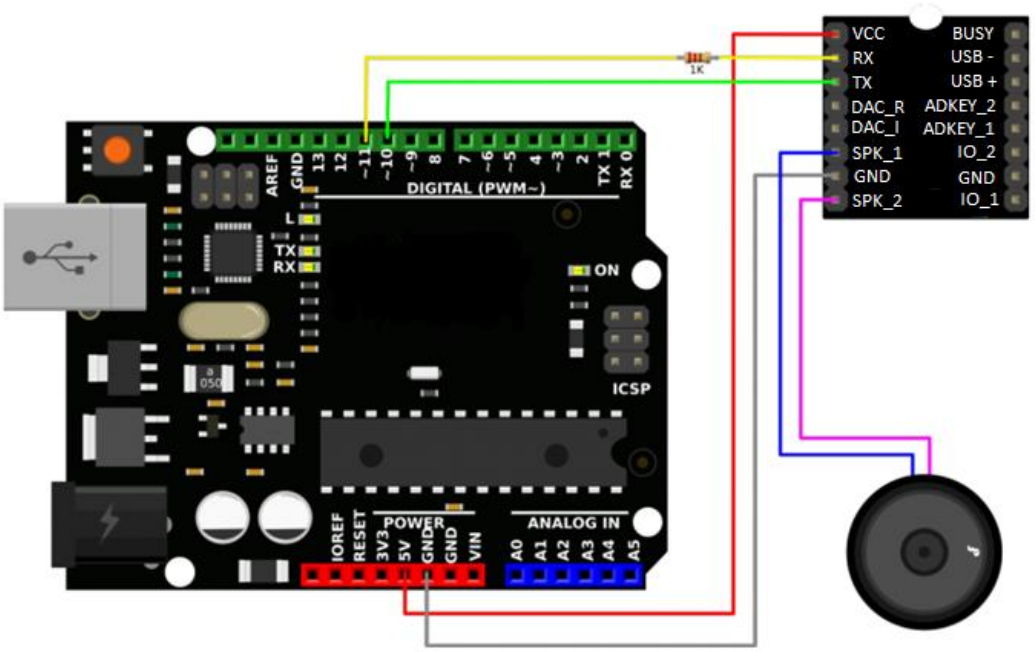


Figura 9: Esquema montaje DFPlayerMini

2.3.3. Funcionamiento

Lo primero que debemos hacer, como ya hemos hecho varias veces, es instalar la librería para usar el DFPlayerMini. En este caso se llama DFRobotDFPlayerMini.

A continuación, escribiremos el siguiente código:

```
#include "Arduino.h"
#include "SoftwareSerial.h"
#include "DFRobotDFPlayerMini.h"

SoftwareSerial mySoftwareSerial(10, 11); // RX, TX
DFRobotDFPlayerMini myDFPlayer;

void setup()
{
  mySoftwareSerial.begin(9600); // velocidad marcada por el
DFPlayerMini
  Serial.begin(9600);

  Serial.println();
  Serial.println(F("Inicializamos DFPlayer ..."));

  if (!myDFPlayer.begin(mySoftwareSerial)) {  //Use softwareSerial to
communicate with mp3.
    Serial.println(F("Error:"));
    Serial.println(F("1.Compruebe la conexión!"));
    Serial.println(F("2.Inserte la tarjeta SD!"));
    while (true);
  }

  Serial.println(F("DFPlayer Mini funcionando."));
  //----Set volume----
  myDFPlayer.volume(10);  //Set volume value (0~30).
```

```
28    //----Set different EQ----
29    myDFPlayer.EQ(DFPLAYER_EQ_NORMAL);
30    //----Set device we use SD as default----
31    myDFPlayer.outputDevice(DFPLAYER_DEVICE_SD);
32  }
33
34  void loop()
35  {
36    for(int i=1;i<=4;i++).
37    {
38      Serial.println(i);
39      myDFPlayer.play(i);
40      delay(6000);
41    }
42  }
```

Una vez que tengamos nuestro Arduino listo para funcionar, vamos a entender el código:

- En la línea 5, definimos un puerto alternativo en los pines 10 y 11, ya que nuestro mini reproductor necesita un puerto serie para comunicarse con Arduino y necesitamos el propio de Arduino para las pruebas.
- En la línea 6, declaramos el reproductor DFPlayerMini.
- En la línea 10, inicializamos el puerto alternativo que hemos declarado.
- En la línea 17, inicializamos el reproductor y a continuación comprobamos si la conexión ha sido correcta.
- En la línea 27, indicamos el volumen (tiene que ser un valor entre 0-30).
- En la línea 29, indicamos el tipo de ecualizador, en nuestro caso vamos a usar *DFPLAYER_EQ_NORMAL*, ya que nuestros sonidos serán simples.
- En la línea 31, indicamos de donde vamos a obtener los ficheros que vamos a reproducir, en nuestro caso una SD.

2.3.4. Problema espada laser 3

En esta sección, solamente tenemos un problema, ya que toda la lógica intermedia esta hecha en las secciones anteriores, por lo tanto, solo nos quedaría integrar la reproducción de sonidos cuando sea necesario.

Para ello hay que tener tres cosas en cuentas:

1- En que estado se encuentra la espada (quieta, moviéndose o impacto).
2- El pin BUSY, que nos indicará si hay algún sonido reproduciéndose.
3- Prioridad de los sonidos, ya que los impactos siempre van a tener una prioridad mayo

Capítulo 3. Resolución de problemas

En este apartado, iremos dando posibles soluciones a los problemas que se han ido planteando en el capítulo 2.

3.1. Solución: Problema interrupciones 1

Usaremos el montaje de la figura 3, pero sin el led, solamente usaremos el botón y mediremos el tiempo para decidir si las pulsaciones son largas o cortas. Para ellos usaremos el siguiente código:

```
#define Tm=500;  //pulsaci*o*n corta es de menos de Tm, larga de m*a*s
de Tm
volatile int flag=0;
volatile int state=0;  //vamos a encender un led mientras
// mantenga pulsado el bot*o*n.
int bpin=2;

void setup() {
  pinMode(bpin,INPUT_PULLUP);
  pinMode(13,OUTPUT);  //el led 13 est*a* intergrado en la placa
  attachInterrupt(INT0,func,CHANGE);
  Serial.begin(9600);
}

void loop() {
  if(flag==1)
  {
      Serial.println("Corta");
      flag=0;
  }
  if(flag==2)
  {
      Serial.println("Larga");
      flag=0;
  }
  digitalWrite(13,state);
}
void func (void)
{
  static unsigned long int tin;
  static unsigned long int tout;
  if(!digitalRead(bpin))
  {
      tin=millis();
      state=1;
  }
  else
  {
      state=0;
      tout=millis();
      if(tout-tin>=Tm)
        flag=2;
      else
        flag=1;
        Serial.print(tout-tin);
        Serial.print(": ");
  }

}
```

Este código tiene varias cosas interesantes que son específicas de las interrupciones y que vamos a ir analizando:

1- En las líneas 4 y 5, vemos que tenemos las variables declaradas como *volatile*, para que puedan ser usadas en la función de interrupción.
2- En la línea 7, declaramos que el pin al que vamos a conectar el botón es el pin 2, que es el pin asociado a la interrupción *INT0*.
3- En la línea 12, inicializamos la interrupción (como hemos visto en el apartado 2.1.3.2).
4- En la línea 13, inicializamos el puerto serie para escribir en él.
5- Dentro de la función *loop()*, solamente comprueba la variable '*flag'* y escribe por el puerto serie dependiendo su valor (líneas 19 y 24). También escribe en el pin 13 (led de la placa de Arduino) el valor de la variable *state*.
6- Por último, tenemos la función de interrupción (*func()*), que su funcionamiento es el siguiente:
   - En las líneas 31 y 32 declaramos dos variables que usaremos para calcular el tiempo que pulsamos el botón.
   - En el '*if*' entraríamos en el primer cambio de nivel, cuando pasamos de HIGH → LOW, y guardamos el valor de la función *millis()*, que es el valor en milisegundos que han pasado desde que comenzó la ejecución del código, en la variable *tin*. En el '*else*' entraríamos cuando dejamos de pulsar el botón, que sería el siguiente cambio de nivel, de LOW → HIGH, y volvemos a guardar el valor de la función *millis()*, pero en la variable *tout*. Una vez que tenemos los dos valores, comprobamos si la diferencia es mayor o igual al tiempo que hemos definido como corto (*Tm*) y así diferenciamos entre pulsación corta o pulsación larga.

3.2. Solución: Problema interrupciones 2

Para la resolución de este problema, el montaje que usaremos será el mismo que hemos usado en la sección 3.1

```
1  /*****************************************************************/
2  /*                       Definiciones .                          */
3  /*****************************************************************/
4
5  #define DEBUG 1 //Si debug es 1 imprimimos resultados intermedios
6  #define Tm 1000  //pulsaci*o*n corta es de menos de Tm, larga de m*a*s
7  de Tm
8
9  /*****************************************************************/
10 /*                       Variables glovales                      */
11 /*****************************************************************/
12
13 volatile int flag=0;   // Nos da el número depulsaciones seg*u*n el
14 enunciado
15 volatile int state=0;  // Va a ser una variable
16 //que nos va a decir si la espada est*a* encendida o no
17 int bpin=2;
18 volatile unsigned long int tin;  // almacenamos cuando se pulsa
19 volatile unsigned long int t;    // almacenamos tiempo actual
20 volatile unsigned long int tout; // almacenamos cuando se suelta
21 volatile int puls=0;             // variable que se pone a 1 la
22 interrupci*o*n se ejecute
23 // (cuado se pulse) y que el loop pondr*a* a cero 1 segundo despues
24 volatile int n=0;    // n*u*mero de pulsaciones
25
26 /*****************************************************************/
27 /*                           Setup .                             */
28 /*****************************************************************/
29 void setup() {
```

```
30    pinMode(bpin,INPUT_PULLUP);
31    pinMode(13,OUTPUT);  //led integrado en la placa
32    attachInterrupt(INT0,func,CHANGE);
33    Serial.begin(9600);
34  }
35  /*********************************************************************/
36  /*                              loop .                               */
37  /*********************************************************************/
38  void loop()
39  {
40    t=millis();
41    if(puls==1)
42    {
43      if(t-tin>=Tm && n==0)  // llevo m*a*s de TM ms pulsado y es la
44  primera pulsaci*o*n
45      {
46        flag=1;
47        puls=0;
48        n=0;
49      }
50    }
51    else if(t-tout>=Tm && n!=0)
52    {
53        if(n>=2)
54           flag=n;
55        n=0;
56        puls=0;
57        if(DEBUG) Serial.println(flag);
58    }
59
60    /*  Que hacemos en funci*o*n del valor de flag*/
61    if(flag==1)
62    {
63        if(state)
64        {
65          Serial.println("Espada apagada");
66          state=0;
67        }
68        else
69        {
70          Serial.println("Espada Encedida");
71          state=1;
72        }
73        flag=0;
74    }
75    if(flag==3)
76    {
77        Serial.println("Cambia de color");
78        flag=0;
79    }
80    if(flag==5)
81    {
82        Serial.println("Cambia de sonido");
83        flag=0;
84    }
85    digitalWrite(13,state);
86  }
87  /*********************************************************************/
88  /*                          Interrupci*o*n
89  */
90  /*********************************************************************/
```

```
91  void func (void)
92  {
93    if(!digitalRead(bpin))
94    {
95        tin=millis();
96        puls=1; // cada vez que pulsemos puls se pone a 1
97    }
98    else if(puls==1)  //si he dejado del pulsar con menos de Tm ms
99    {
100       tout=millis();  // almaceno cuando he dejado de pulsar
101       n++;
102       puls=0;// ponemos puls a cero
103   }
104 }
```

En este caso, explicaremos el funcionamiento del código en general.

En primer lugar, vamos a explicar el funcionamiento de la función de interrupción.

Al igual que antes, leemos el valor del pin al que tenemos conectado el botón, y si lo acabamos del pulsar, guardamos el valor de la función *millis()* en la variable *tin* y ponemos la variable *puls* a 1. Si por el contrario, lo que acabamos de hacer es soltar el botón, guardamos el valor de la función *millis()* en la variable *tout*, incrementamos en 1 la variable *n* y ponemos la variable *puls* a 0.

La función *loop()* la explicaremos más detenidamente, para comprenderla bien:

1- En la línea 40, guardamos el valor de la función *millis()*.
2- En la línea 41, comprobamos si el botón está pulsado (*puls == 1*). Una vez dentro, comprobamos si el tiempo que llevamos pulsando el botón es mayor o igual a *Tm* (*t-tin == Tm*) y que no hemos hecho varias pulsaciones (línea 43). Cuando se cumplen estas condiciones, cambiamos el valor de las variables.
3- En la línea 51, comprobamos si la diferencia entre el tiempo actual y la última pulsación es mayor o igual a *Tm* (*t-tout >= Tm*) y si el número de pulsaciones es distinto de 0. Una vez dentro, comprobamos si las veces que hemos pulsado el botón es mayor o igual a 2 (línea 53) y guardamos ese valor en la variable *flag*.
4- A continuación, dependiendo del valor de la variable *flag* y *state*, tomamos las decisiones adecuadas.

3.3. Solución: Problema espada laser 1

Una vez hayamos realizado y entendido los dos problemas anteriores, solucionar este es sencillo, solamente tenemos que crear algunas funciones para trabajar con la tira de leds.

En este caso, hemos dividido el condigo en funciones, para que sea más legible y fácil de entender.

- Principal: contiene el *setup()* y el *loop()*.

```
/*****************************************************************/
/*                        Librerias .                            */
/*****************************************************************/

#include <Adafruit_NeoPixel.h>

/*****************************************************************/
/*                        Definiciones .                         */
/*****************************************************************/

#define DEBUG 1 //Si debug es 1 imprimimos resultados intermedios
#define Tm 500  //pulsaci*o*n corta es de menos de Tm, larga de m*a*s
de Tm
#define NUMLEDS 2    // n*u*mero de leds (o de grupo de leds)
#define DATAPIN 6     // pin de datos de al tira de leds
#define NUMCOLORS 8  // numero de colores de la espada predefinidos
#define B_PIN 2           // pin al que est*a* conectado el pulsador

/*****************************************************************/
/*                        Variables glovales                     */
/*****************************************************************/

volatile int flag=0;             // Nos da el n*u*mero depulsaciones
seg*u*n el enunciado
volatile int state=0;            // state=1 espada encendida, state=0
apagada
volatile unsigned long int tin;  // almacenamos cuando se pulsa
volatile unsigned long int t;    // almacenamos tiempo actual
volatile unsigned long int tout; // almacenamos cuando se suelta
volatile int puls=0;             // variable que se pone a 1 la
interrupci*o*n se ejecute
// (cuado se pulse) y que el loop pondr*a* a cero Tm milisegundo
despu*e*s
volatile int n=0;    // n*u*mero de pulsaciones
int r = 0;                       // variables para c*o*digo RBG de
iluminaci*o*n
int g = 0;
int b = 255;
int r_imp = 0;                   // variables para c*o*digo RBG de
impacto
int g_imp = 255;
int b_imp = 0;
// Colores predefinidos iluminaci*o*n: 0-Blue,  1-Red,   2-Green,   3-
Magenta, 4-Bromn, 5-Cornsilk
// Colores predefinidos impacto      : 0-Green, 1-Green, 2-Magenta, 3-
Green,   4-White,  5-Green
int j=0;    // almacena el color de la espada
Adafruit_NeoPixel pixel = Adafruit_NeoPixel(NUMLEDS,DATAPIN,NEO_RGB +
NEO_KHZ800); // variable para acceder a los leds
```

```
/*****************************************************************/
/*                            Setup .                            */
/*****************************************************************/

void setup() {
  pinMode(B_PIN,INPUT_PULLUP);
  pinMode(13,OUTPUT);  //led integrado en la placa
  attachInterrupt(INT0,func_bot,CHANGE);
  Serial.begin(9600);
  pixel.begin();
  leds_off();
}

/*****************************************************************/
/*                            loop .                             */
/*****************************************************************/

void loop()
{
  flag=boton();  // Nos devuelve el n*u*mero de pulsaciones segun el
enunciado

  /*  Que hacemos en funci*o*n del valor de flag*/
  if(flag==1)
  {
      if(state)
      {
        Serial.println("Espada apagada");
        state=0;
        leds_off();
      }
      else
      {
        Serial.println("Espada Encedida");
        state=1;
        leds_on();
      }
      flag=0;
  }
  if(flag==3 && state==1)
  {
      Serial.println("Cambia de color");
      flag=0;
      change_color();
  }
  if(flag==5 && state==1)
  {
      Serial.println("Cambia de sonido");
      flag=0;
  }
  digitalWrite(13,state); // El led integrado en la placa nos dice si
*e*sta est*a* encendida
}
```

Como podemos observar, la estructura es similar a la del código de la sección 2.1.3.5. Podemos ver algunas variables más que usaremos para el cambio de color cuando se produzca un impacto (ver sección X.Y.Z) y dentro de *loop()*, se van usando funciones que explicaremos a continuación.

- *boton()*:

```
int boton()
{
  t=millis();
  if(puls==1)               // Se ha pulsado el boton; tin guarda
cuando se puls*o*
  {
    if(t-tin>=Tm && n==0)   // si llevo m*a*s de TM ms pulsado y es la
primera pulsaci*o*n
    {
      flag=1;
      puls=0;               // hago puls=0 para que no siga contando en
n
      n=0;
    }
  }
  else if(t-tout>=Tm && n!=0)
  {
      if(n>=2)
         flag=n;
      n=0;
      puls=0;
      if(DEBUG) Serial.println(flag);
  }
  return flag;  // Nos devuelve el numero de pulsaciones
}
```

Esta función calcula el valor de la variable *flag* en función del tiempo que hemos mantenido el botón pulsado o el número de veces que hemos pulsado el botón.

- *func_boton()*: función de interrupción.

```
/*********************************************************************/
/* Interrupci*o*n. Se ejecuta cuando el pin bpin cambia de estado
*/
/*********************************************************************/
void func_bot (void)
{

  if(!digitalRead(B_PIN)) // Si se ha pulsado el boton
  {
      tin=millis();
      puls=1;           // cada vez que pulsemos puls se pone a 1
  }
  else if(puls==1)     //si he dejado del pulsar y puls sigue a 1, con
menos de Tm ms
  {
      tout=millis();  // almaceno cuando he dejado de pulsar
      n++;            // incremento la cuenta de n
      puls=0;         // ponemos puls a cero
  }
```

Esta función es exactamente igual que la que analizamos en la sección anterior. Obtiene el tiempo en el que pulsamos y en el que dejamos de pulsar para calcular la duración y también cuenta el número de veces que pulsamos de forma rápida y continua.

- *change_color()*:

```
void change_color()
{
  j=j+1;
  if(j>=NUMCOLORS)
    j=0;
  get_color();
  for ( int i = 0; i < NUMLEDS; i++)
  {
    pixel.setPixelColor(i,r,g,b);
  }
  pixel.show();
  Serial.println(j);
}
```

Esta función cambia el color completo de la espada según haya cambiado la función *get_color()*, que veremos en el siguiente punto, lo valores de las variables *r*,*g* y *b*, que dependen de la variable *j*. El valor de la variable *j*, va aumentando en uno hasta que llegue a *NUMCOLORS*, que volvería otra vez a cero.

- *get_color()*:

```
void get_color(){
  switch (j) {
    case 0: // Blue - Green
      r = 0;
      g = 0;
      b = 255;
      r_imp = 0;
      g_imp = 255;
      b_imp = 0;
      break;
    case 1: // Red - Green
      r = 255;
      g = 0;
      b = 0;
      r_imp = 0;
      g_imp = 255;
      b_imp = 0;
      break;
    case 2: // Green - Magenta
      r = 0;
      g = 255;
      b = 0;
      r_imp = 229;
      g_imp = 9;
      b_imp = 127;
      break;
    case 3: // Magenta - Green
      r = 229;
      g = 9;
      b = 127;
      r_imp = 0;
      g_imp = 255;
      b_imp = 0;
      break;
```

```
    case 4: // Bromn - White
      r = 150;
      g = 75;
      b = 0;
      r_imp = 255;
      g_imp = 255;
      b_imp = 255;
      break;
    case 5: // Cornsilk - Green
      r = 255;
      g = 248;
      b = 220;
      r_imp = 0;
      g_imp = 255;
      b_imp = 0;
      break;
    default: // se apagan todos los leds
      r = 0;
      g = 0;
      b = 0;
      r_imp = 0;
      g_imp = 0;
      b_imp = 0;
      break;
  }
}
```

Como hemos visto en la función *change_color()*, esta función cambia el valor de las variables *r*,*g* y *b*, que representan el color de la espada , y de *r_imp*, *g_imp* y *b_imp*, que serán los colores de la espada cuando impacta. Estos valores se calculan según el valor que tenga la variable *j*.

- *leds_on()*:

```
void leds_on()
{
  Serial.println("Encendemos los leds");
  int i;
  if(NUMLEDS % 2 == 0)
        i = NUMLEDS/2;
  else
        i = NUMLEDS/2 + 0.5;

  for ( int j = 0; j <= i; j++)
  {
    pixel.setPixelColor(j,r,g,b);
      pixel.setPixelColor(NUMLEDS - j,r,g,b);
    pixel.show();
    delay(30);
  }
}
```

Esta función enciende los leds cuando encendemos la espada. Ha diferencia de como encendemos los leds en la función *change_color()*, aquí lo que hacemos es ir encendiendo los leds desde la parte baja de la espada hasta la parte alta, y teniendo en cuenta que la tira de leds hará una forma de 'U', vamos encendiendo los leds desde el primero hacia adelante y desde el último hacia atrás.

- *leds_off()*:

```
void leds_off()
{
  Serial.println("Apagamos los leds");
  int i;
  if(NUMLEDS % 2 == 0)
        i = NUMLEDS/2;
  else
        i = NUMLEDS/2 + 0.5;

  for ( int j = 0; j <= i; j++)
  {
    pixel.setPixelColor(i + j,0,0,0);
      pixel.setPixelColor(i - j,0,0,0);
    pixel.show();
    delay(30);
  }
}
```

Esta función apaga los leds cuando apagamos la espada. Seguimos la misma idea que para encenderlos, pero, al contrario, vamos apagando desde el led central hacia los dos extremos.

En definitiva, este primer código general, contiene el funcionamiento del botón y la tira de leds, que es el siguiente:

- Cuando mantenemos el botón pulsado más de *Tm* milisegundos, encendemos o apagamos la espada, dependiendo del estado en el que se encuentre en ese momento, y hacemos lo necesario con los leds.
- En caso de pulsar 3 veces el botón, cambiamos el color de la espada.
- En caso de pulsar 5 veces el botón, cambiamos los sonidos de la espada (ver sección X.Y.Z).

3.4. Solución: Problema cálculo umbrales

Usaremos el siguiente código para poder determinar dichos umbrales:

```
#include <I2Cdev.h>
#include "MPU6050.h"
#include "Wire.h"

const int mpuAddress = 0x68;  // Puede ser 0x68 o 0x69
MPU6050 mpu(mpuAddress);

int16_t ax, ay, az;
int16_t gx, gy, gz;

// Factores de conversion
const float accScale = 16.0 * 9.81 / 32768.0;
const float gyroScale = 250.0 / 32768.0;
```

```
14  void setup()
15  {
16     Serial.begin(9600);
17     Wire.begin();
18     mpu.initialize();
19     mpu.setFullScaleAccelRange(MPU6050_ACCEL_FS_16) ; // Establecemos
20  el m*a*ximo rango de medida de las aceleraciones
21     Serial.println(mpu.testConnection() ? F("IMU iniciado
22  correctamente") : F("Error al iniciar IMU"));
23  }
24
25  void loop()
26  {
27    vel_ace();
28    delay(10);
29  }
30  void vel_ace()
31  {
32    float acel,rot;
33    static float acelf=10,rotf=0;
34    float alfa=0.5;  // un peque*n*o filtro para el ruino
35    mpu.getAcceleration(&ax, &ay, &az);
36    mpu.getRotation(&gx, &gy, &gz);
37    acel=(abs(ax)+abs(ay)+abs(az))*accScale;
38    acel=abs(acel);
39    rot=(abs(gx)+abs(gy)+abs(gz))*gyroScale;
40    rot=abs(rot);
41    acelf=alfa*acelf+(1-alfa)*acel;  // esto es un filtro, no hace falta
42    rotf=alfa*rotf+(1-alfa)*rot;     // esto es un filtro, no hace falta
43    Serial.print("rotf: ");
44    Serial.println(rotf);
45  }
```

En este código vemos algunas cosas diferentes:

- En la línea 19, cambiamos el rango máximo de medida de la aceleración a 16, por lo que tenemos que cambiar también la variable *accScale* (línea 12).
- La función *vel_ace()* es la que hace todo el trabajo:
  1- Calculamos los valores de los 6 ejes (líneas 35 y 36), aceleración y velocidad angular.
  2- Calculamos los valores absolutos de dichas medidas (líneas 37-40) y sus valores filtrados (líneas 41 y 42).
  3- Pintamos dichos valores (debemos cambiar las variables a pintar, *rotf* o *acelf*, dependiendo de lo que queramos ver en ese momento).

Si queremos ver estos valores gráficamente, necesitamos usar *Serial Ploter*: *Herramientas* > *Serial Ploter*.

Las gráficas que obtenemos son las siguiente:

- Para la velocidad tenemos la siguiente.

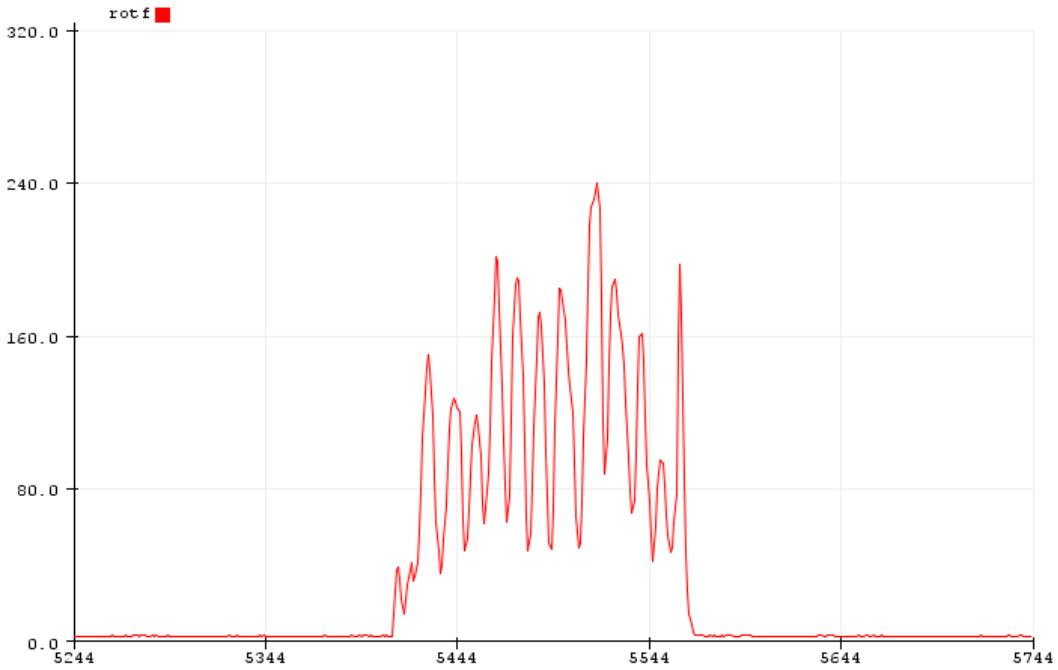


Figura 10: Umbral velocidad

Como podemos observar, se puede poner un umbral de 60-80 para detectar cuando la espada se está moviendo. Pero debemos tener en cuenta los movimientos de rotación, que sacaría la siguiente figura con un movimiento mínimo.

Figura 11: Umbral velocidad error

Por lo tanto, para verificar que es un movimiento relevante, vamos a calcular una aceleración mínima de movimiento.

- Para calcular los umbrales de aceleración sacamos las siguientes gráficas:

Figura 12: Umbral aceleración movimiento

Como vemos aquí, para los movimientos relevantes, podemos poner un umbral de movimiento de 20, y así nos quitamos los movimientos pequeños de rotación.

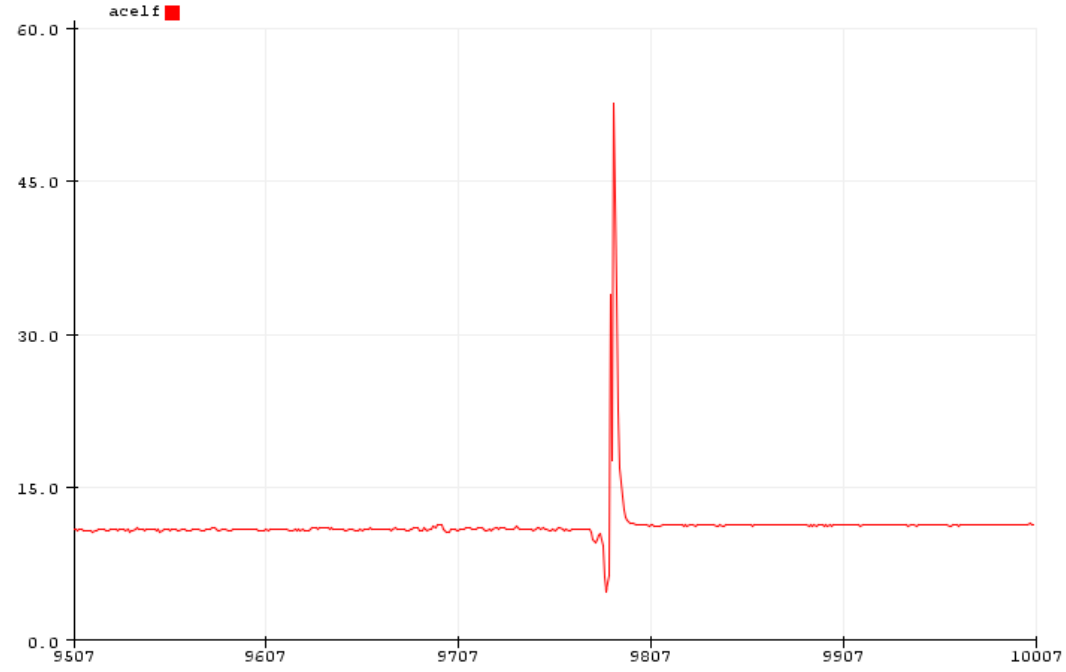


Figura 13: Umbral aceleración impacto

Para ver el umbral del impacto es más sencillo, ya que, como podemos observar, se genera un pico bastante grande, ya que un impacto tiene una gran aceleración. Podemos definir el umbral de aceleración de impacto con un valor de 40.

3.5. Solución: Problema movimiento o impacto

En esta solución, partiremos del código anterior:

```
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"

#define U_acel_imp 40 // Umbral de la aceleraci*o*n para impacto
#define U_acel_mov 15 // Umbral de la aceleraci*o*n para movimiento
#define U_rot 20 // Umbral de la velocidad

const int mpuAddress = 0x68; // Puede ser 0x68 o 0x69
MPU6050 mpu(mpuAddress);

int16_t ax, ay, az;
int16_t gx, gy, gz;
const float accScale = 16.0 * 9.81 / 32768.0;
const float gyroScale = 250.0 / 32768.0;

void setup(){
  Serial.begin(9600);
  Wire.begin();
  mpu.initialize();
  mpu.setFullScaleAccelRange(MPU6050_ACCEL_FS_16);
  Serial.println(mpu.testConnection() ? F("IMU iniciado
correctamente") : F("Error al inciar IMU"));
  pinMode(13,OUTPUT);  // El led integrado en la placa
}

void loop(){
  int res=vel_ace();
  if(res==1){
    digitalWrite(13,HIGH);
    Serial.println("Movimieto");
  }else if(res==0){
    digitalWrite(13,LOW);
    Serial.println("Quieto");
  }else if(res==2)
    Serial.println("Impacto");
  delay(10);
}

int vel_ace(){
  int res=0;   // 0 no se mueve, 1 se mueve, 2 golpe
  float acel,rot;
  static float acelf=10,rotf=0;
  float alfa=0.5;  // un peque*n*o filtro para el ruino
  mpu.getAcceleration(&ax, &ay, &ax);
  mpu.getRotation(&gx, &gy, &gz);
  acel=(abs(ax)+abs(ay)+abs(az))*accScale;
  acel=abs(acel);
  rot=(abs(gx)+abs(gy)+abs(gz))*gyroScale;
  rot=abs(rot);
  acelf=alfa*acelf+(1-alfa)*acel;  // esto es un filtro, no hace falta
  rotf=alfa*rotf+(1-alfa)*rot;     // esto es un filtro, no hace falta
  if (rotf >= U_rot & acelf >= U_acel_mov)
    res=1;
  if (acelf >= U_acel_imp)
    res=2;
return res;}
```

Como se ha dicho al principio de la sección, nos hemos basado en el código de la sección 3.4, y se han añadido los cambios necesarios para completar el nuevo problema:

- En las líneas 5, 6 y 7, definimos los umbrales que hemos calculado anteriormente.
- En el *loop()*, dependiendo del valor que nos devuelva la función *vel_ace()* (0, 1 o 2) vemos en qué situación está la espada, si en movimiento, quieta o a sufrido un impacto.
- En la función *vel_ace()*,una vez que tenemos lo valores de la velocidad y la aceleración, teniendo en cuanta los valores de los umbrales, tomamos la decisión del estado de la espada (0 → quieta, 1 → en movimiento, 2 → impacto).

3.6. Solución: Problema espada laser 2

Al igual que en el apartado 3.3, dividiremos el código en funciones (como nos basamos en el mismo código, solo pondremos las funciones nuevas y las que hayan sufrido alguna modificación) para que se entienda mejor y sea más fácil de comprender.

- Principal: contiene el *setup()* y el *loop()*.

```
/*****************************************************************/
/*                       Librerias .                             */
/*****************************************************************/
#include <Adafruit_NeoPixel.h>

#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"
#include "Arduino.h"
#include "SoftwareSerial.h"

/*****************************************************************/
/*                       Definiciones .                          */
/*****************************************************************/

#define DEBUG 1           //Si debug es 1 imprimimos resultados
intermedios
#define Tm 500            //pulsaci*o*n corta es de menos de Tm, larga
de m*a*s de Tm
#define NUMLEDS 11       // n*u*mero de leds (o de grupo de leds)
#define DATAPIN 6         // pin de datos de al tira de leds
#define NUMCOLORS 6       // numero de colores de la espada
predefinidos
#define U_acel 40         // Umbral de la aceleraci*o*n
#define U_rot 20          // Umbral de la velocidad
#define B_PIN 2           // pin al que est*a* conectado el pulsador

/*****************************************************************/
/*                       Variables glovales                      */
/*****************************************************************/
/*    PULSADOR      */
volatile int flag=0;             // Nos da el n*u*mero depulsaciones
seg*u*n el enunciado
volatile int state=0;            // state=1 espada encendida, state=0
apagada
volatile unsigned long int tin;  // almacenamos cuando se pulsa
volatile unsigned long int t;    // almacenamos tiempo actual
volatile unsigned long int tout; // almacenamos cuando se suelta
volatile int puls=0;             // variable que se pone a 1 la
interrupci*o*n se ejecute
                                 // (cuado se pulse) y que el loop
pondr*a* a cero Tm milisegundo despu*e*s
volatile int n=0;                // n*u*mero de pulsaciones
```

```
/*    LEDS     */
int r = 0;                          // variables para c*o*digo RBG de
iluminaci*o*n
int g = 0;
int b = 255;
int r_imp = 0;                      // variables para c*o*digo RBG de
impacto
int g_imp = 255;
int b_imp = 0;
// Colores predefinidos iluminaci*o*n: 0-Blue,  1-Red,   2-Green,   3-
Magenta, 4-Bromn, 5-Cornsilk
// Colores predefinidos impacto      : 0-Green, 1-Green, 2-Magenta, 3-
Green,   4-White,  5-Green
int j=0;    // almacena el color de la espada
Adafruit_NeoPixel pixel = Adafruit_NeoPixel(NUMLEDS,DATAPIN,NEO_RGB +
NEO_KHZ800); // variable para acceder a los leds
/*    IMU    */
const int mpuAddress = 0x68;  // Puede ser 0x68 o 0x69
MPU6050 mpu(mpuAddress);
int16_t ax, ay, az;
int16_t gx, gy, gz;
// Factores de conversion
const float accScale = 16.0 * 9.81 / 32768.0;   // Cambaimos el 2 por
16
const float gyroScale = 250.0 / 32768.0;
/*****************************************************************/
/*                            Setup .                            */
/*****************************************************************/

void setup() {
  pinMode(B_PIN,INPUT_PULLUP);
  pinMode(13,OUTPUT);  //led integrado en la placa
  attachInterrupt(INT0,func_bot,CHANGE);
  Serial.begin(9600);

  pixel.begin();
  leds_off();
  /*   IMU    */
  Wire.begin();
  mpu.initialize();
  mpu.setFullScaleAccelRange(MPU6050_ACCEL_FS_16); // Establecemos el
m*a*ximo rango de medida de las aceleraciones
  Serial.println(mpu.testConnection() ? F("IMU iniciado
correctamente") : F("Error al iniciar IMU"));
}

/*****************************************************************/
/*                            loop .                             */
/*****************************************************************/

void loop()
{
  flag=boton();  // Nos devuelve el n*u*mero de pulsaciones segun el
enunciado
  if(state){
    int res = vel_ace();
    if(res == 2){
      impacto();
      Serial.println("Impacto");
    } else if (res == 1){
      Serial.println("Movimiento");
```

```
    } else {
      Serial.println("Quieto");
    }
  }
  /*  Que hacemos en funci*o*n del valor de flag*/
  if(flag==1) {
      if(state)
      {
        Serial.println("Espada apagada");
        state=0;
        leds_off();
      }
      else
      {
        Serial.println("Espada Encedida");
        state=1;
        leds_on();
      }
      flag=0;
  } else if(flag==3 && state) {
      Serial.println("Cambia de color");
      flag=0;
      change_color();
  } else if(flag==5 && state) {
      Serial.println("Cambia de sonido");
      flag=0;
  }
  digitalWrite(13,state); // El led integrado en la placa nos dice si
*e*sta est*a* encendida
}
```

En esta parte del código, no se ha modificado nada, solamente se han añadido las cosas necesarias para ver la situación de la espada:

- En la primera parte, se han incluido las librerías, se han añadido las definiciones de los umbrales y se han declarado las variables necesarias para el cálculo de la velocidad y la aceleración.
- En el *setup(),* se ha inicializado el MPU-6050 para poder trabajar con él.
- En el *loop()*, siempre que la espada esta encendida (*state = 1*), llamamos a la función *vel_ace()* y vemos la situación de la espada para ver qué decisión tomamos.

- *impacto()*:

```
void impacto()
{
  for ( int i = 0; i < NUMLEDS; i++)
  {
    pixel.setPixelColor(i,r_imp,g_imp,b_imp);
  }
  pixel.show();
  delay(150);
  for ( int i = 0; i < NUMLEDS; i++)
  {
    pixel.setPixelColor(i,r,g,b);
  }
  pixel.show();
}
```

Esta función hace el efecto de iluminación del impacto. Para ellos, sigue los siguientes pasos:

1- Guarda en todos los leds el color del impacto, pero sin cambiarlo (*pixel.setPixelColor(i, r_imp, g_imp, b_imp)*).
2- Ilumina todos los leds, ya con el color cambiado al color de impacto (*pixel.show()*).
3- Esperamos un tiempo para que se perciba el efecto (*delay(150)*).
4- Volvemos al color de iluminación de la misma forma. Guardamos en todos los leds el color de iluminación, pero sin cambiarlo (*pixel.setPixelColor(i, r, g, b)*).
5- Ilumina todos los leds con el color de iluminación (*pixel.show()*).

- *vel_ace()*:

```
int vel_ace()
{
  int res=0;   // 0 no se mueve, 1 se mueve, 2 golpe
  float acel,rot;
  static float acelf=10,rotf=0;
  float alfa=0.5;  // un peque*n*o filtro para el ruino
  mpu.getAcceleration(&ax, &ay, &az);
  mpu.getRotation(&gx, &gy, &gz);
  acel=(abs(ax)+abs(ay)+abs(az))*accScale;
  acel=abs(acel);
  rot=(abs(gx)+abs(gy)+abs(gz))*gyroScale;
  rot=abs(rot);
  acelf=alfa*acelf+(1-alfa)*acel;  // esto es un filtro, no hace falta
  rotf=alfa*rotf+(1-alfa)*rot;     // esto es un filtro, no hace falta
  if (rotf >= U_rot & acelf >= U_acel_mov)
    res=1;
  if (acelf >= U_acel_imp)
    res=2;
return res;
}
```

Esta función es igual a la que hemos visto en el apartado 3.5, nos devuelve la situación de la espada dependiendo del valor calculado de la velocidad angula y la aceleración, y de los umbrales que hemos definido.

3.7. Solución: Problema espada laser 3

Como hemos hecho en las secciones 3.3 y 3.6, dividiremos el código en funciones para que sea más entendible y lo podamos explicar mejor

- Principal: contiene el *setup()* y el *loop()*.

```
/*****************************************************************/
/*                       Librerias .                             */
/*****************************************************************/

#include <Adafruit_NeoPixel.h>
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"
#include "Arduino.h"
#include "SoftwareSerial.h"
#include "DFRobotDFPlayerMini.h"

/*****************************************************************/
/*                       Definiciones .                          */
/*****************************************************************/
```

```
#define DEBUG 1           //Si debug es 1 imprimimos resultados
intermedios
#define Tm 500            //pulsaci*o*n corta es de menos de Tm, larga
de m*a*s de Tm
#define NUMLEDS 288       // n*u*mero de leds (o de grupo de leds)
#define DATAPIN 6         // pin de datos de al tira de leds
#define NUMCOLORS 6       // numero de colores de la espada
predefinidos
#define NUMSONG 3         // numero de sonidos de la espada
predefinidos
#define U_acel_imp 40     // Umbral de la aceleraci*o*n para impacto
#define U_acel_mov 15     // Umbral de la aceleraci*o*n para
movimiento
#define U_rot 20          // Umbral de la velocidad
#define B_PIN 2           // pin al que est*a* conectado el pulsador
#define S_PIN 3           // pin al que est*a* conectado la bandera de
sonido del DFPlayerMini

/*****************************************************************/
/*                      Variables glovales                       */
/*****************************************************************/

volatile int flag=0;             // Nos da el n*u*mero depulsaciones
seg*u*n el enunciado
volatile int state=0;            // state=1 espada encendida, state=0
apagada
volatile unsigned long int tin;  // almacenamos cuando se pulsa
volatile unsigned long int t;    // almacenamos tiempo actual
volatile unsigned long int tout; // almacenamos cuando se suelta
volatile int puls=0;             // variable que se pone a 1 la
interrupci*o*n se ejecute
                                 // (cuado se pulse) y que el loop
pondr*a* a cero Tm milisegundo despu*e*s
volatile int n=0;                // n*u*mero de pulsaciones
/*    LEDS     */
int r = 0;                       // variables para c*o*digo RBG de
iluminaci*o*n
int g = 0;
int b = 255;
int r_imp = 0;                   // variables para c*o*digo RBG de
impacto
int g_imp = 255;
int b_imp = 0;
// Colores predefinidos iluminaci*o*n: 0-Blue,  1-Red,   2-Green,   3-
Magenta, 4-Bromn, 5-Cornsilk
// Colores predefinidos impacto      : 0-Green, 1-Green, 2-Magenta, 3-
Green,   4-White,  5-Green
int j=0;    // almacena el color de la espada
Adafruit_NeoPixel pixel = Adafruit_NeoPixel(NUMLEDS,DATAPIN,NEO_RGB +
NEO_KHZ800); // variable para acceder a los leds
/*    IMU    */
const int mpuAddress = 0x68;  // Puede ser 0x68 o 0x69
MPU6050 mpu(mpuAddress);
int16_t ax, ay, az;
int16_t gx, gy, gz;
// Factores de conversion
const float accScale = 16.0 * 9.81 / 32768.0;   // Cambaimos el 2 por
16
const float gyroScale = 250.0 / 32768.0;
/*       SONIDO       */
SoftwareSerial mySoftwareSerial(10, 11); // RX, TX
```

```
DFRobotDFPlayerMini myDFPlayer;
// Sonidos predefinidos movimiento   : 0 -> 5, 1 -> 7, 2 -> 9
// Sonidos predefinidos impacto      : 0 -> 4, 1 -> 6, 2 -> 8
int s = 0; // Almacena el sonido de la espada
int songSleep = 1;
int songON = 2;
int songOFF = 3;
int songImp = 4;
int songMov = 5;
int stateSong = 0;      // 0 = quieto, 1 = movimiento, 2 = impacto
int stateSongOld = 0;   // 0 = quieto, 1 = movimiento, 2 = impacto
/*****************************************************************/
/*                            Setup .                            */
/*****************************************************************/
void setup() {
  mySoftwareSerial.begin(9600);
  pinMode(B_PIN,INPUT_PULLUP);
  pinMode(S_PIN,INPUT);
  pinMode(13,OUTPUT);  //led integrado en la placa
  attachInterrupt(INT0,func_bot,CHANGE);
  Serial.begin(9600);
  Serial.println(F("Initializing DFPlayer ... (May take 3~5
seconds)"));
  if (!myDFPlayer.begin(mySoftwareSerial)) {  //Use softwareSerial to
communicate with mp3.
    Serial.println(F("Unable to begin:"));
    Serial.println(F("1.Please recheck the connection!"));
    Serial.println(F("2.Please insert the SD card!"));
    while (true);
  }
  Serial.println(F("DFPlayer Mini online."));
  //----Set volume----
  myDFPlayer.volume(20);  //Set volume value (0~30)
  //----Set different EQ----
  myDFPlayer.EQ(DFPLAYER_EQ_NORMAL);
  //----Set device we use SD as default----
  myDFPlayer.outputDevice(DFPLAYER_DEVICE_SD);
  pixel.begin();
  leds_off();
  /*   IMU    */
  Wire.begin();
  mpu.initialize();
  mpu.setFullScaleAccelRange(MPU6050_ACCEL_FS_16); // Establecemos el
m*a*ximo rango de medida de las aceleraciones
  Serial.println(mpu.testConnection() ? F("IMU iniciado
correctamente") : F("Error al iniciar IMU"));
}
/*****************************************************************/
/*                            loop .                             */
/*****************************************************************/
void loop()
{
  flag=boton();  // Nos devuelve el n*u*mero de pulsaciones segun el
enunciado
  if(state){
    stateSongOld = stateSong;
    int pinBusy = digitalRead(S_PIN); // 0 = musica reproduciendo 1 =
musica parada
    int res = vel_ace();
    if(res == 2){
      Serial.println("Impacto");
```

```
      stateSong = 2;
      impacto(true);
      myDFPlayer.play(songImp);
      delay(300);
      while(digitalRead(S_PIN) == 0); // Dejamos que se reproduzca el
sonido por completo
      impacto(false);
    } else if (res == 1){
      if((pinBusy == 1 || stateSongOld == 0) && stateSongOld != 2){
        stateSong = 1;
        Serial.println("Movimiento");
        myDFPlayer.play(songMov);
        delay(500);
      }
    } else {
      if(pinBusy == 1){
        stateSong = 0;
        Serial.println("Activo musica quieto");
        myDFPlayer.play(songSleep);
        delay(500);
      }
    }
  }
  /*  Que hacemos en funci*o*n del valor de flag*/
  if(flag==1)
  {
      if(state)
      {
        Serial.println("Espada apagada");
        state=0;
        myDFPlayer.play(songOFF);
        leds_off();
      }
      else
      {
        Serial.println("Espada Encedida");
        state=1;
        myDFPlayer.play(songON);
        leds_on();
        delay(500);
        while(digitalRead(S_PIN) == 0); // Dejamos que se reproduzca
el sonido por completo
      }
      flag=0;
  }
  if(flag==3 && state)
  {
      Serial.println("Cambia de color");
      flag=0;
      change_color();
  }
  if(flag==5 && state)
  {
      Serial.println("Cambia de sonido");
      flag=0;
         change_song();
  }
  digitalWrite(13,state); // El led integrado en la placa nos dice si
*e*sta est*a* encendida
}
```

De la primera parte no explicaremos nada, ya que con los comentarios y con lo que hemos visto durante todo el desarrollo se puede entender perfectamente.

Centraremos nuestra explicación en el *loop()*:

- En primer lugar, guardaremos el estado antiguo del sonido (*stateSongOld = stateSong*), leemos el estado del pin BUSY y leemos el resultado de la función *vel_ace()*.
- Una vez tenemos esto valores, vemos que decisión tomar:
  1- Si se ha producido un imparto (*res = 2*), cambiamos la variable *stateSong* a 2, llamamos a la función *impacto(true),* para que encienda los leds con el color establecido para el impacto, reproducimos el sonido de impacto (*myDFPlayer.play(songImp)*), esperamos 300 ms o hasta que se termine de reproducir el archivo de sonido (para que sea visible el cambio de color del impacto) y luego volvemos a poner los leds en su color (*impacto(false)*).
  2- Si estamos en movimiento (*res = 1*), tenemos que comprobar que no vengamos justo de un impacto (*stateSongOld != 2*), porque no se podría reproducir un sonido de movimiento de forma inminente, y que no se esté reproduciendo nada (*pinBusy = 1*) o se esté reproduciendo el sonido de 'quieto' (*stateSongOld == 0*), que sería menos prioritario. Si se cumplen las condiciones anteriores, reproducimos el sonido de movimiento que este en se momento (*myDFPlayer.play(songMov)*) y establecemos un *delay* de 500ms para dejar trabajar al DFPlayerMini y no mandar acciones que puedan ser contradictorias.
  3- Si estamos quietos, solo reproduciríamos el sonido de 'quieto' si no se está reproduciendo nada en ese momento (*pinBusy == 1*).
- Con respecto a las pulsaciones del botón, si se enciende o se apaga la espada, reproducimos el sonido que corresponda (en el caso del encendido, dejamos que se reproduzca el sonido completo) y si pulsado el botón 5 veces, modificamos las variables de sonido llamando a la función *change_song()*.

- *boton()*:

```
int boton()
{
  t=millis();
  if(puls==1)                // Se ha pulsado el boton; tin guarda
cuando se puls*o*
  {
    if(t-tin>=Tm && n==0)   // si llevo m*a*s de TM ms pulsado y es la
primera pulsaci*o*n
    {
      flag=1;
      puls=0;                // hago puls=0 para que no siga contando en
n
      n=0;
    }
  }
  else if(t-tout>=Tm && n!=0)
  {
      if(n>=2)
         flag=n;
      n=0;
      puls=0;
      if(DEBUG) Serial.println(flag);
  }
  return flag;  // Nos devuelve el numero de pulsaciones
}
```

- *func_boton()*: función de interrupción.

```
/*********************************************************************/
/* Interrupci*o*n. Se ejecuta cuando el pin bpin cambia de estado
*/
/*********************************************************************/
void func_bot (void)
{

  if(!digitalRead(B_PIN)) // Si se ha pulsado el boton
  {
      tin=millis();
      puls=1;           // cada vez que pulsemos puls se pone a 1
  }
  else if(puls==1)     //si he dejado del pulsar y puls sigue a 1, con
menos de Tm ms
  {
      tout=millis();  // almaceno cuando he dejado de pulsar
      n++;            // incremento la cuenta de n
      puls=0;         // ponemos puls a cero
  }
}
```

- *change_color()*:

```
void change_color()
{
  j=j+1;
  if(j>=NUMCOLORS)
    j=0;
  get_color();
  for ( int i = 0; i < NUMLEDS; i++)
  {
    pixel.setPixelColor(i,r,g,b);
  }
  pixel.show();
  Serial.println(j);
}
```

- *get_color()*:

```
void get_color(){
  switch (j) {
    case 0: // Blue - Green
      r = 0;
      g = 0;
      b = 255;
      r_imp = 0;
      g_imp = 255;
      b_imp = 0;
      break;
    case 1: // Red - Green
      r = 255;
      g = 0;
      b = 0;
      r_imp = 0;
      g_imp = 255;
      b_imp = 0;
      break;
    case 2: // Green - Magenta
      r = 0;
```

```
      g = 255;
      b = 0;
      r_imp = 229;
      g_imp = 9;
      b_imp = 127;
      break;
    case 3: // Magenta - Green
      r = 229;
      g = 9;
      b = 127;
      r_imp = 0;
      g_imp = 255;
      b_imp = 0;
      break;
    case 4: // Bromn - White
      r = 150;
      g = 75;
      b = 0;
      r_imp = 255;
      g_imp = 255;
      b_imp = 255;
      break;
    case 5: // Cornsilk - Green
      r = 255;
      g = 248;
      b = 220;
      r_imp = 0;
      g_imp = 255;
      b_imp = 0;
      break;
    default: // se apagan todos los leds
      r = 0;
      g = 0;
      b = 0;
      r_imp = 0;
      g_imp = 0;
      b_imp = 0;
      break;
  }
}
```

- *leds_on()*:

```
void leds_on()
{
  Serial.println("Encendemos los leds");
  int i;
  if(NUMLEDS % 2 == 0)
        i = NUMLEDS/2;
  else
        i = NUMLEDS/2 + 0.5;

  for ( int j = 0; j <= i; j++)
  {
    pixel.setPixelColor(j,r,g,b);
    pixel.setPixelColor(NUMLEDS - j,r,g,b);
    pixel.show();
    delay(30);
  }
}
```

- *leds_off()*:

```
void leds_off()
{
  Serial.println("Apagamos los leds");
  int i;
  if(NUMLEDS % 2 == 0)
        i = NUMLEDS/2;
  else
        i = NUMLEDS/2 + 0.5;

  for ( int j = 0; j <= i; j++)
  {
    pixel.setPixelColor(i + j,0,0,0);
    pixel.setPixelColor(i - j,0,0,0);
    pixel.show();
    delay(30);
  }
}
```

- *impacto()*:

```
void impacto(boolean on)
{
  if(on){
    for ( int i = 0; i < NUMLEDS; i++)
    {
      pixel.setPixelColor(i,g_imp,r_imp,b_imp);
    }
  } else {
    for ( int i = 0; i < NUMLEDS; i++)
    {
      pixel.setPixelColor(i,g,r,b);
    }
  }
  pixel.show();
}
```

En este caso, hemos modificado un poco la función. Le pasamos una variable para indicar si se encienden los leds con el color 'impacto' o queremos que se iluminen con el color 'quieto', ya que no pondremos la espada en color 'quieto' hasta que no termine el sonido del impacto.

- *vel_ace()*:

```
int vel_ace()
{
  int res=0;   // 0 no se mueve, 1 se mueve, 2 golpe
  float acel,rot;
  static float acelf=10,rotf=0;
  float alfa=0.5;  // un peque*n*o filtro para el ruino
  mpu.getAcceleration(&ax, &ay, &az);
  mpu.getRotation(&gx, &gy, &gz);
  acel=(abs(ax)+abs(ay)+abs(az))*accScale;
  acel=abs(acel);
  rot=(abs(gx)+abs(gy)+abs(gz))*gyroScale;
  rot=abs(rot);
  acelf=alfa*acelf+(1-alfa)*acel;  // esto es un filtro, no hace falta
  rotf=alfa*rotf+(1-alfa)*rot;     // esto es un filtro, no hace falta
  if (rotf >= U_rot & acelf >= U_acel_mov)
```

```
    res=1;
  if (acelf >= U_acel_imp)
    res=2;
return res;
}
```

- *change_song()*:

```
void change_song()
{
  s=s+1;
  if(s>=NUMSONG)
    s=0;
  get_song();
  Serial.println(s);
}
```

Esta función es similar a *change_solor()* pero en vez de cambiar los colores de reposo e impacto, lo que cambia, es el sonido de la espada cuando esta en movimiento y cuando sufre un impacto.

- *get_color()*:

```
void get_song()
{
switch (s) {
    case 0: // 4 - 5
      songImp = 4;
      songMov = 5;
      break;
    case 1: // 6 - 7
      songImp = 6;
      songMov = 7;
      break;
    case 2: // 8 - 9
      songImp = 8;
      songMov = 9;
      break;
    default: // Error
      songImp = 10;
      songMov = 10;
      break;
  }
}
```

Esta función asigna los valores a las variables *songImp* y *songMov* en función al valor de *s,* y eso determina los sonidos de impacto y de movimiento.

Capítulo 4. Conclusiones

En un trabajo didáctico, su principal característica debe ser su claridad para abordar conceptos completamente nuevos para los usuarios, por eso hemos tratado de dividir todas las partes y explicarlas una por una, para luego más tarde ir uniendo todos los problemas que se han ido planteando a lo largo del trabajo.

Para hacer más comprensible el proyecto, lo hemos dividido en tres bloques principales.

En el primer bloque, hemos tratado temas más básicos, como las funciones principales de Arduino o como trabajar con leds o botones. También hemos desarrollado conceptos más complicados como las funciones, pero sin entrar en detalles más complejos, ya que tratamos de hacer una didáctica para personas que están comenzando.

En el segundo bloque, hemos introducido uno de los módulos, el MPU-6050, con el que su parte más compleja no ha sido la conexión con el propio Arduino, si no el cálculo que hemos tenido que hacer con los valores que leíamos de él.

En el tercer bloque, hemos usado otro módulo un poco más complejo, como es el DFPlayerMini. En este caso, aparte de la complejidad que tenía la programación, como ha sido la configuración de un nuevo puerto serie, debíamos tener cuidado con las conexiones entre el Arduino y el módulo, ya que es un componente delicado y debemos tener cuidado con los voltajes de alimentación.

Por último, en la parte final del trabajo, hemos dedicado una sección a desarrollar posibles soluciones a todos los problemas que hemos ido proponiendo, aunque no son soluciones únicas, ya que hay otras alternativas que podrían ser igual de válidas.

Capítulo 5. Anexos

En este capítulo, trataremos dos temas que quedarían fuera de la didáctica de inicialización en Arduino, la programación y la electrónica, pero son bastante útiles para el diseño final y la construcción de este.

5.1. Anexo I: Consumo

Este apartado no es muy largo, pero interesante, porque nos determinará el tamaño de la fuente de alimentación que vamos a necesitar, dependiendo del consumo, en amperios, de nuestro circuito.

Para este proyecto, hemos dividido la medición del consumo en dos partes:

1- En la primera, vamos a ver cuanto consume nuestra tira de leds, que es el componente que más consumo tendrá. Para ello, tenemos que poner los leds a máximo rendimiento, que serían cuando los leds se encienden en color blanco, y medir su consumo en ese momento. Nosotros, tras hacer esa prueba hemos determinado que el consumo máximo es de 2.5A, aunque el consumo medio, con los colores que hemos configurado es de 1.5A.
2- En segundo lugar, tenemos que medir el consumo del resto del circuito. La mayor dificultad que existe para hacer esta medida de forma exacta es poner el circuito a máximo rendimiento durante un periodo de tiempo. Para ello, hemos puesto el volumen del altavoz al máximo y hemos generados varios impactos con el MPU6050, y hemos sacado las medidas de esos instantes. Tras varias pruebas, he valor medio que hemos sacado es de 0,4A.

Con estos dos datos, sacamos que el consumo máximo del circuito es de 2.9A.

Con esta característica y teniendo en cuenta que tanto el circuito como los demás componentes se pueden alimentar entre 3.3V y 5V, hemos elegido, como fuente de alimentación, las pilas recargables 18650 de 3.7V y 3200mAh.

5.2. Anexo II: Diseño de placa de circuito impreso

Otro parte interesante, es el diseño de nuestra placa de circuito impreso, más conocido con las siglas en inglés PCB (Printed Circuit Board).

Para ello, debemos tener en cuenta, que el único sitio donde puede ir insertada es en el mango de la espada, por lo que esto nos limitará su tamaño.

Nosotros hemos tenido en cuenta los siguientes puntos para el diseño de nuestra PCB.

1- Como ya hemos comentado, el más importante es su ubicación, por lo que hemos tratado de hacerla lo más estrecha posible.
2- Los leds y el botón no van insertados en la placa, por lo que usaremos pines para luego soldarlos con cables.
3- El altavoz va directamente soldado al DFPlayerMini, por lo que no ocupa lugar en esta placa.
4- Colocar todos los componentes en una capa. Esto lo hacemos, por si queremos añadir las baterías por el otro lado de la placa y que quede todo compacto.

Con todas estas condiciones que nos hemos puesto nosotros mismos, hemos diseñado el siguiente PCB.

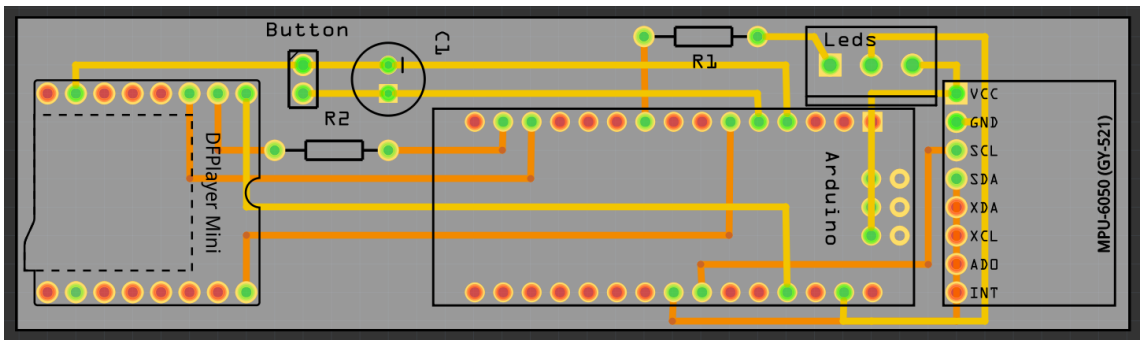


Figura 14: Diseño PCB

Bibliografía

1. https://es.wikipedia.org/wiki/Arduino_Uno [En línea]
2. https://programarfacil.com/blog/breve-introduccion-al-arduino/ [En línea]
3. http://diwo.bq.com/descubre-el-pulsador/ [En línea]
4. https://github.com/adafruit/Adafruit_NeoPixel [En línea]
5. https://naylampmechatronics.com/blog/45_tutorial-mpu6050-acelerometro-y-giroscopio.html [En línea]
6. https://www.luisllamas.es/arduino-mp3-dfplayer-mini/ [En línea]
7. https://wiki.dfrobot.com/DFPlayer_Mini_SKU_DFR0299 [En línea]
8. https://github.com/DFRobot/DFRobotDFPlayerMini [En línea]