Informe Arduino

Alfonso José Mateos Hoyos (DNI: 44059172G)
Manuel Salgado de la Iglesia (DNI: 70925985B)
Francisco Pinto Santos (DNI: 70918455)

Puesto de trabajo número 9

Práctica I

I. LED parpadeante

Para realizar el montaje de este circuito, comenzamos midiendo la resistencia, para asegurarnos que era de 330 ohmios, tras lo cual, montamos el LED y conectamos la tierra a la resistencia.
Para probar el LED, primero lo conectamos a la entrada de 5V, y al ver que se encendía, terminamos el montaje conectándolo al pin digital número13.

Una vez hecho el montaje procedimos a programar el funcionamiento con el IDE que nos proporciona Arduino.
Quizás, esta fue la parte más difícil, pues no teníamos mucha experiencia con ello, y tuvimos que documentarnos por internet, pero nos ayudó bastante que la sintaxis fuera C, pues tenemos bastante experiencia en este lenguaje.

En cuanto al código, como se puede ver en la Figura 1.1.2, consistió en declarar unas constantes (pin de conexión y el delay entre parpadeos), declarar el pin al que conectamos el positivo del LED como salida, y por último, el funcionamiento en sí.
Este consistió en encender el LED, esperar, apagarlo y volver a esperar. Lo cual se ejecutaba de continuo al estar en la función loop.

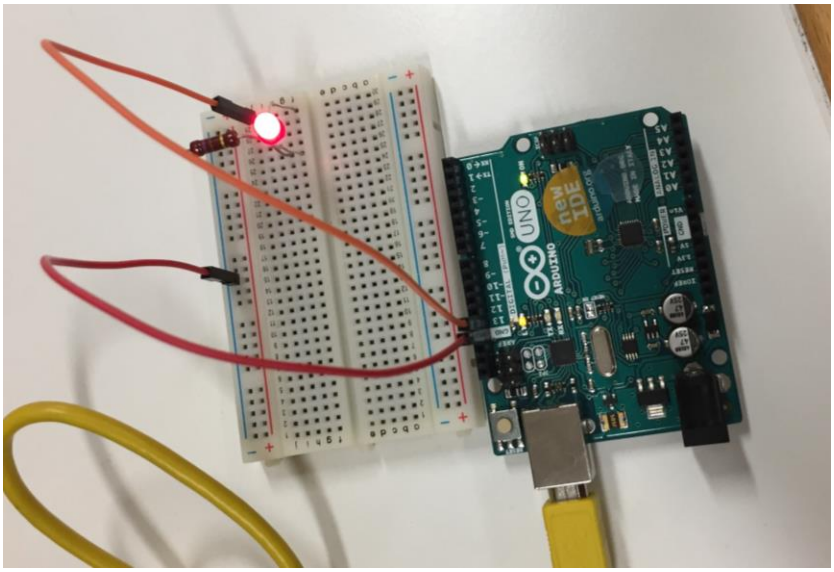

Figura 1.1.1 Montaje del circuito

```
//Constantes:
//  + pin en el que hemos conectado la alimentacion positiva del led
//  + duracion del intervalo entre 2 parpadeos
#define MYPIN 13
#define DELAY 500

void setup() {
  //declarar el pin como salida
  pinMode(MYPIN,OUTPUT);
}

void loop() {
  //activar la salida MYPIN (encender el LED)
  digitalWrite(MYPIN,HIGH);
  //esperar
  delay(DELAY);
  //desactivar la salida MYPIN (apagar el LED)
  digitalWrite(MYPIN,LOW);
  //esperar
  delay(DELAY);
}
```

Figura 1.1.2 Código correspondiente a este montaje

II. Circuito detector de luz/oscuridad

Este segundo ejercicio consiste en repetir el circuito realizado en la sesión anterior, un detector de luz/oscuridad, esta vez en Arduino, para lo cual reutilizamos el montaje del apartado anterior.

Para el montaje utilizamos una resistencia de 330Ω, (para el LED) y 1kΩ, las cuales medimos para corroborar que los valores y el funcionamiento de estos eran correctos y así evitar cualquier error.
Tras ello caracterizamos el LDR mediante el Serial Plotter para conocer los valores umbrales de este y poder realizar el programa después.

Montamos el circuito, conectando la parte positiva del LED al pin digital 9 y la otra a la resistencia de 330Ωla cual, iba conectada a tierra. Por otra parte, montamos un divisor de tensión con el LDR y la resistencia de 1kΩ, los cuales conectaos a las entradas de 5V y tierra. Tras lo cual conectamos a un pin analógico el punto intermedio entre el LDR y la resistencia, pues al tratarse de un divisor de tensión, según vaya cambiado la luz, en el LDR variara el valor de su resistencia, y por tanto cambia el voltaje medido en ese punto.

Una vez hecho el montaje, programamos el código (Figura 1.2.1.) necesario para hacer que el LED se encienda o apague cuando pongamos el dedo encima del LDR.

Declaramos los pines necesarios (pin analógico del divisor de tensión como input y pin del LED como output), además, iniciamos la terminal a 9600 baudiospara poder ver la variación de los valores del LDR para la posterior depuración si fuera necesaria.

Por último, en el loop, hicimos un analogRead(LDRinput)para leer el voltaje en el punto medio del divisor de tensión (si ponemos o quitamos el dedo sobre el LDR, como este es sensible a la luz, varia el valor de la resistencia y por tanto del voltaje en el punto medio del divisor), y con este valor haremos dos sentencias condicionales ifelse para hacer que el LED se encienda cuando el valor medido sea inferior a un umbral (lo que significa que no hay luz), mediante digitalWrite(LEDpin, LOW/HIGH). Podríamos hacer que se encendiera o se apagara al poner el dedo encima, simplemente invirtiendo las sentencias.

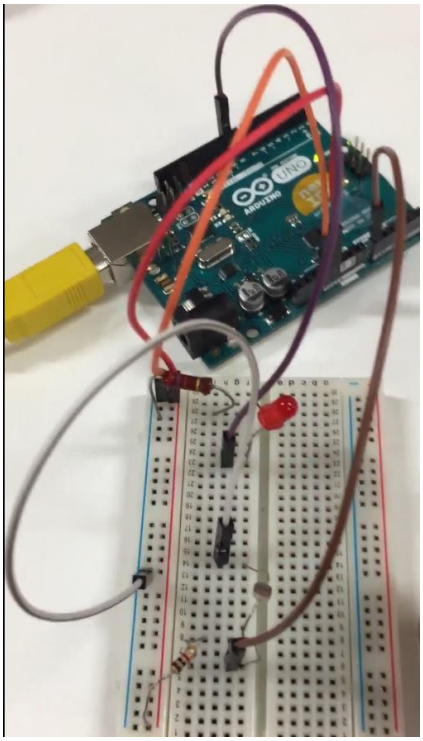

Figura 1.2.1 Circuito detector luz/oscuridad

```
//Constantes:
//  + pin en el que hemos conectado el LDR
//  + pin en el que hemos conectado el LED
//  + limite a partir del cual el LDR detecta que
//          no hay luz (sacado a ensallo y error)
//  + delay entre mediciones del LDR y actuacion
//          sobre el LED
#define LDRINPUT A0
#define LEDPIN 9
#define LDRLIMIT 990
#define DELAY 500

void setup() {
  //declarar el pin del LDR como entrada (buscamos medir de este)
  //   y el del LED como salida (para ilumniarlo)
  pinMode(LDRINPUT,INPUT);
  pinMode(LEDPIN,OUTPUT);
  //Iniciar la terminal a 9600 baudios
  Serial.begin(9600);
}

void loop() {
  //Leer la entrada del pin del LDR
  int output = analogRead(LDRINPUT);
  //mostrar el dato leido
  Serial.println(output);

  //si hay luz (aumenta el valor de la entrada del LDR) apagamos el LED
  //    en caso contrario lo encendemos
  if( output > LDRLIMIT )
    digitalWrite(LEDPIN,LOW);
  else
    digitalWrite(LEDPIN,HIGH);

  //esperar DELAY ms
   delay(DELAY);
}
```

Figura 1.2.2 Código detector luz/oscuridad

III. Coche fantástico

En este tercer ejercicio se nos pedía montar “el coche fantástico”, el cual consiste en hacer ver que una luz pasa de izquierda a derecha y viceversa por los LED.
Cabe destacar que como no nos dio tiempo en el laboratorio a terminar el código, realizamos el montaje en casa y lo probamos ahí con la excepción de que era un Arduino megarev 2560 en vez de un Arduino uno.

Para el montaje necesitamos ocho LED y sus ocho resistencias correspondientes (de 330Ω), los cuales probamos y medimos respectivamente para evitar errores desconocidos.

La parte con polaridad positiva de los LED que se encuentran en los extremos se conectó a los pines 12 y 13 de la parte digital, respectivamente, mientras que la del resto de los LED se conectó a los seis pines de la parte analógica. En cuanto a la parte con polaridad negativa de cada LED, esta se conectó a las resistencias. Posteriormente, cada resistencia se conectó a tierra.

Una vez realizado el montaje, comenzamos la explicación del programa.
En primer lugar, vemos (Figuras 1.3.2.x) que tenemos una serie de constantes:

- número de LED a encender y por tanto, número de pines
- pines digitales en los que hemos conectado los LED de los extremos
- máximo valor que puede recibir la función analogWrite, para aplicar un voltaje proporcional a su posición a cada LED analógico.
- valor base del analogWrite para que todos los LED se mantengan encendidos.

Posteriormente, se declaró un vector con los pines a los cuales hemos conectado los LED y los establecimos en modo salida.

Ahora pasamos a la parte que se irá repitiendo (*loop*). Lo primero es hacer un bucle para que “se propague el haz de luz” de izquierda a derecha (Figura 1.3.2.2). Para ello lo que se hace es apagar todos los LED que se tenga a la derecha del actual, se atenúan los de la izquierda del mismo para dar esa sensación de propagación y se enciende al máximo en el que estamos.

Más tarde (principio de la Figura 1.2.3.3), se hace la transición del haz de luz viajando hacia la derecha a emitirse hacia la izquierda fuera del bucle, consiguiendo así una mayor fluidez.

Por último, se hace el bucle que se ha realizado anteriormente para encender los LED de izquierda a derecha de manera invertida, consiguiendo el efecto contrario, es decir, que se enciendan de derecha a izquierda.

Al final de la Figura 1.9 tenemos la función que se ha utilizado para escribir en los pines dependiendo de si el pin en cuestión es digital o analógico.

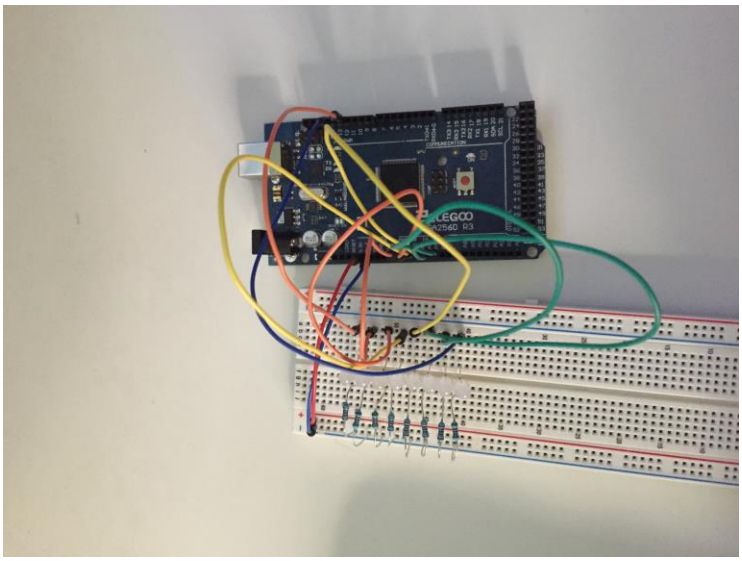

Figura 1.3.1 Montaje coche fantástico

```
//Constantes:
//  + numero de pins que hay
//  + pin donde hemos conectado el led de la izda
//  + pin donde hemos conectado el led de la dcha
//  + maximo valor que acepta analogWrite
//  + minimo valor para darle a un LED conectado a
//       un pin analogico
//  + delay de refresco de los LED
#define NPINS 8
#define DIGITAL1 12
#define DIGITAL2 13
#define MAXV 255
#define BASEV 45
#define DELAYTIME 100

//Vector que contiene los pins en los que estan conectados los LED
int pins[] = { DIGITAL1, A0, A1, A2, A3, A4, A5, DIGITAL2};

//Function para escribir sobre un pin, indiferentemente si es analogico o digital
void writeAnalogicOrDigital(int pin, int analogicValue, int digitalValue);

void setup() {
  //declarar todos los pins contenidos en pins como salida
  int i;
  for(i+0; i<NPINS; i++)
    pinMode(pins[i],OUTPUT);
  //iniciar el terminal a 9600 baudios
  Serial.begin(9600);
}
```

Figura 1.3.2.1 Código coche fantástico. Parte 1.

```
void loop() {
  int i, current, currentV;

 //Encender de forma gradual de izquierda a derecha
 for(current=0; current<NPINS; current++){

      //apagar desde el led actual hacia la derecha
      for(i=0; i<current ;i++){
          currentV = BASEV + (MAXV/(1+current-i));
          writeAnalogicOrDigital(pins[i],currentV,LOW);
      }

      //encender de formga gradual desde el led actual
      //   hacia la izquierda
      for(i=current+1; i<NPINS; i++){
          writeAnalogicOrDigital(pins[i],0,LOW);
      }

      //encender a el maximo el led actual
      writeAnalogicOrDigital(pins[current],MAXV,HIGH);

      //para una mayor suavidad en las transiciones
      if(current < 3)
        digitalWrite(DIGITAL1,HIGH);

      //esperar a la siguiente actualizacion de los LED
      delay(DELAYTIME);
 }

 //para una mayor suavidad en las transiciones
 analogWrite(A4,0);
```

Figura 1.3.2.2 Código coche fantástico. Parte 2.

```
  //para una mayor suavidad en las transiciones
  analogWrite(A4,0);
  digitalWrite(DIGITAL2,HIGH);
  delay(DELAYTIME);
  analogWrite(A5,0);
  digitalWrite(DIGITAL2,HIGH);
  delay(DELAYTIME);
  analogWrite(A5,MAXV);
  digitalWrite(DIGITAL2,HIGH);
  delay(DELAYTIME);

  //Encender de forma gradual de dercha a izquierda
  for(current=NPINS-1; current>=0; current--){

      //apagar desde el led actual hacia la izquierda
      for(i=current-1;i>=0;i--){
         currentV = BASEV + (MAXV/(1+current-i));
         writeAnalogicOrDigital(pins[i],currentV,LOW);
      }

      //encender de formga gradual desde el led actual
      //   hacia la derecha
      for(i=NPINS-1; i>current; i--){
          writeAnalogicOrDigital(pins[i],0,LOW);
      }

      //encender a el maximo el led actual
      writeAnalogicOrDigital(pins[current],MAXV,HIGH);
```

Figura 1.3.2.3 Código coche fantástico. Parte 3.

```
      //para una mayor suavidad en las transiciones
      if(current < 3)
        digitalWrite(DIGITAL1,HIGH);

      //esperar a la siguiente actualizacion de los LED
      delay(DELAYTIME);
  }

}

//Escribe el pin sobre el que se va a escribir, y en el caso de ser
//   digital escribe en el con digitalWrite. En caso contrario, escribe
//   con analogWrite.
void writeAnalogicOrDigital(int pin, int analogicValue, int digitalValue){

  Serial.println(pin);
    if(pin==DIGITAL1 || pin == DIGITAL2)
        digitalWrite(pin,digitalValue);
    else
        analogWrite(pin,analogicValue);
}
```

Figura 1.3.2.4 Código coche fantástico. Parte 4.

IV. Referencias

- https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/
- https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/
- https://www.arduino.cc/reference/en/language/functions/digital-io/digitalread/
- https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/
- https://www.robotshop.com/community/tutorials/show/arduino-5-minute-tutorials-lesson-2-basic-code-amp-blink-led
- https://www.arduino.cc/en/Serial/Println

Práctica II

I. Medir distancia con HC-SR04:

El primer sensor que usaremos será el HC-SR04, un sensor de ultrasonidos, el cual nos permitirá medir distancias.
Su funcionamiento es el siguiente:

- Envía un pulso de alta frecuencia no audible por el ser humano.
- Este pulso rebota en objetos cercanos y es reflejado hacia el sensor, el cual dispone de un micrófono para esta frecuencia.
- Se mide el tiempo entre pulsos, conociendo la velocidad del sonido, y estimamos la distancia del objeto contra el que ha rebotado.

En concreto, el rango de medición teórico de este sensor es de 2cm a 400cm, con una resolución (margen de error) de 0.3cm, aunque en la práctica estos valores están mucho más limitados.

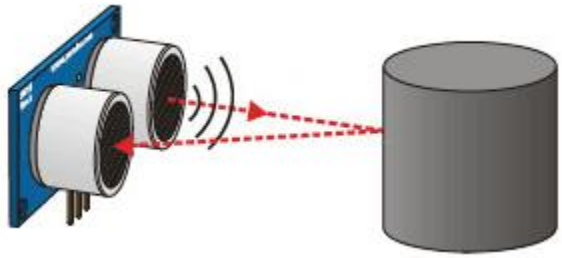

Figura 2.1.1: HC-SR04

La referencia temporal que existe desde que lanzamos una ráfaga hasta que esta aparece en el pin de salida, es debido a que la señal que se actualiza es eléctrica y es más rápida que la ultrasónica.

Dicho todo esto, montamos el circuito tal y como vemos en la Figura 2.1.2, siendo solo necesario el sensor, alimentación y tierra, y dos pines digitales (trigger – output, echo pulse – input). Además, preparamos el código necesario en Arduino. En este código hemos usado una biblioteca que encapsula el uso del sensor. Por ello, solo hemos tenido que llamar a la función que aparece en la línea 13 de la Figura 2.1.3, la cual se encargará de declarar ambos pines, también tendremos que inicializar el terminal serie. Por último, en el loop, simplemente mediremos la distancia que está recogiendo el sensor con measureDistanceCm() y la mostraremos por pantalla.

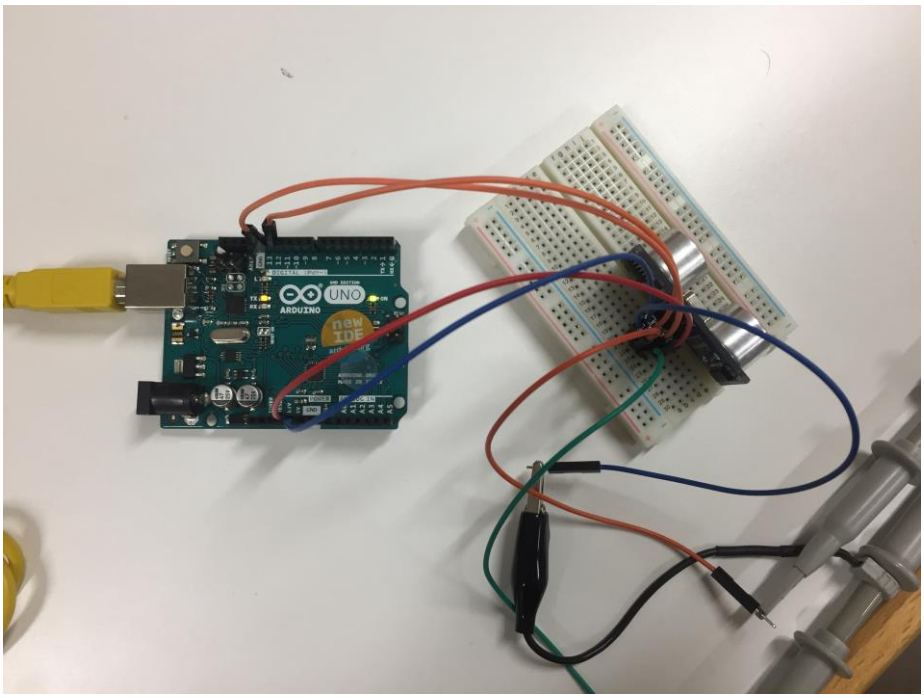

Figura 2.1.2: Circuito HC-SR04

```
//Biblioteca que encapsula el uso del sensor: https://github.com/Martinsos/arduino-lib-hc-sr04
#include <HCSR04.h>

//Constantes:
//  + pin de trigger del HCSR04
//  + pin de echo del HCSR04
//  + delay entre mediciones
#define TRIGGERPIN  13
#define ECHOPIN 12
#define DELAY 100

//Objeto que encpapsula el uso del sensor
UltraSonicDistanceSensor distanceSensor(TRIGGERPIN, ECHOPIN);

void setup () {
    //inicializacion de la terminal serie
    Serial.begin(9600);
}

void loop () {
    //obtener distancia
    double distance = distanceSensor.measureDistanceCm();
    Serial.println(distance);
    //esperar DELAY ms
    delay(DELAY);
}
```

Figura 2.1.3: Código HC-SR04 Arduino

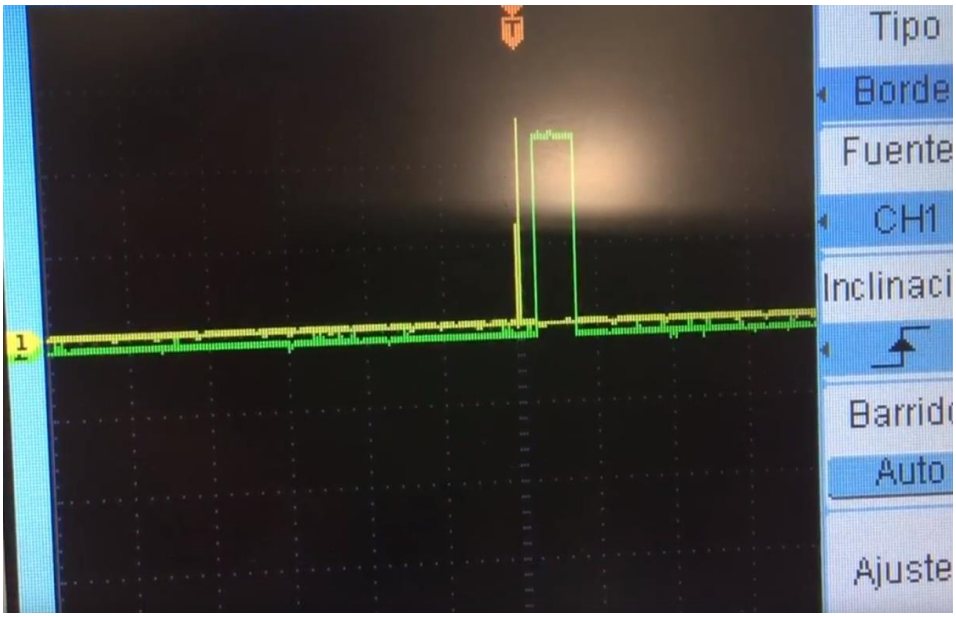

Figura 2.1.4: Señal visualizada en el osciloscopio

II. Sensor de flexión:

Para realizar el montaje de este circuito, comenzamos probando la resistencia (ya que, si no era del valor adecuado, el divisor de tensión no funcionaría adecuadamente), y observamos que era de unos 2.4 kΩ, que era el valor deseado.
Tras ello, procedimos a montar el circuito como venía descrito en la Figura aportada en el informe, resultando en el montaje que se puede aprecia en la Figura 2.2.1.

Una vez hecho el montaje procedimos a programar lo necesario para poder medir los cambios en tensión ocasionados por el sensor.
Esto se puede ver en la Figura 2.2.1, donde se ve el código que realizamos, el cual era muy básico:

- Realizar una medición analógica, que nos daría un valor entre 0 y 255.
- Imprimir esa medición
- Esperar 100 ms (para no realizar mediciones de continuo, sino dejando un pequeño intervalo entre mediciones).

Con esto, podríamos observar los cambios en la tensión según la flexión del sensor.

En cuanto a los resultados, los podemos ver en la Figura 2.2.3 (extraída del vídeo de demostración), que según se dobla el sensor hacia una dirección u otra, el voltaje aumenta o disminuye.
Esto se debe a que el sensor actúa como una resistencia, la cual hemos integrado en un divisor de tensión, en el cual, si realizamos mediciones de voltaje entre las dos resistencias, se puede obtener un valor más alto o bajo dependiendo del voltaje aportado por la fuente (5 voltios en este caso) y el valor de las resistencias.
Por tanto, al doblar el sensor, estamos variando el valor de una de las resistencias del divisor de tensión, haciendo que se obtenga más o menos voltaje en la medición.

Debido a esto, si interpretamos la grafica obtenida como un conjunto de valores entre 0V y 5V en vez de valores entre 0 y 255, estaremos obteniendo la tensión de medición real.
Del mismo modo, conociendo el valor del voltaje (obtenido en cada medición) y el de la otra resistencia (2.4 kΩ), podremos conocer el valor real de resistencia que ofrece el sensor (de 2.5kΩ a 12.5kΩ) aplicando la fórmula de la Figura 2.2.4 y pudiendo así interpretar la escala como los valores de resistencia que ofrece el sensor.

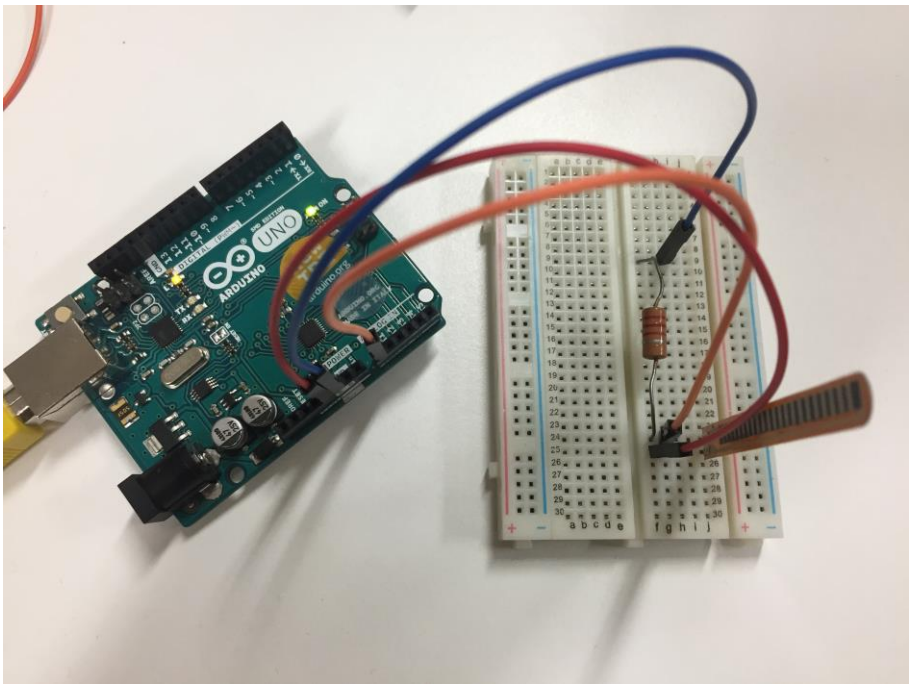

Figura 2.2.1 Montaje del circuito

```
//Constantes:
//  + pin de input del sensor de flexion
//  + delay entre mediciones
#define INPUTPIN A0
#define DELAY 100

void setup() {
  //inicializacion de la terminal serie
  Serial.begin(9600);
}

void loop() {
  //obtener voltaje
  int medicion = analogRead(INPUTPIN);
  //imprimir distancia por puerto serie: para hacer plotting o leerlo
  Serial.println(medicion);
  //esperar DELAY ms
  delay(DELAY);
}
```

Figura 2.2.2 Código correspondiente a este montaje

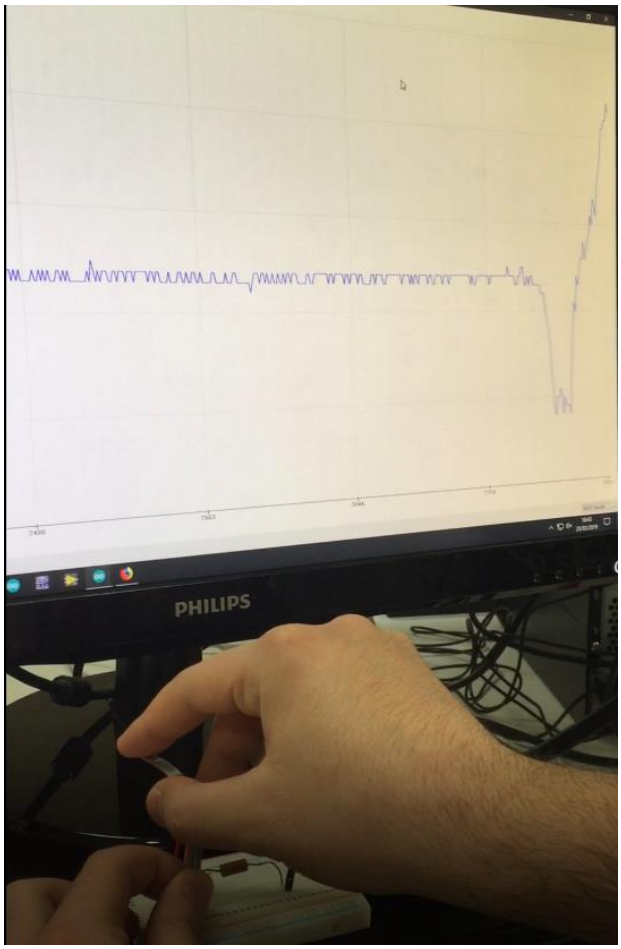

Figura 2.2.3: funcionamiento del sensor

$$V_{\text{out}} = \frac{R_2}{R_1 + R_2} \cdot V_{\text{in}}$$

Figura 2.2.4 Fórmula de la cual despejando R2, se puede obtener la resistencia ofrecida por el sensor de flexión.

III. Sensor de temperatura TMP36:

Por último, se nos pedía trabajar con el sensor de temperatura TMP36.
Para ello, se realizó el montaje tal y como se muestra en la Figura 2.3.1, conectando los tres pines del sensor de temperatura, a tierra, a la alimentación (5V) y a un pin analógico que nos serviría para conseguir las mediciones de temperatura.

Una vez hecho el montaje, pasamos a realizar el código (Figura 2.3.2).
En él se guarda el valor que se ha obtenido a través del pin analógico mencionado antes y se convierte ese valor a través de una fórmula a grados Celsius, como se nos pedía en el enunciado. Mediante esta fórmula se consigue convertir la medida del sensor a voltaje y posteriormente este voltaje pasarlo a grados Celsius.

Esta medición se realizaría en bucle (*loop*) con el objetivo de obtener una gráfica en la cual se mostrase como la temperatura iba aumentando o disminuyendo, como vemos en la Figura 2.3.3, la cual es una captura del vídeo realizado del funcionamiento del sensor, en el cual cogemos el sensor con los dedos de forma que aumente su temperatura o lo soltamos para que disminuya.

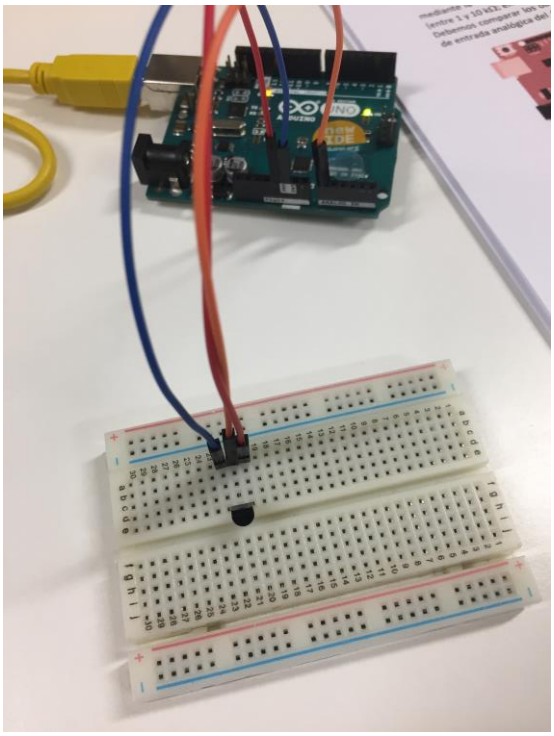

Figura 2.3.1: Montaje del circuito.

```
//Constantes:
//  + pin de input del sensor de flexion
//  + delay entre mediciones
#define INPUTPIN A0
#define DELAY 100

void setup() {
  //inicializacion de la terminal serie
  Serial.begin(9600);
}

void loop() {
  //obtener voltaje
  int medicion = analogRead(INPUTPIN);
  //convertir medicion de voltaje a grados celsius
  float celsius = (medicion*5.0/1024 - 0.5)*100;
  //imprimir distancia por puerto serie: para hacer plotting o leerlo
  Serial.println(celsius);
  //esperar DELAY ms
  delay(DELAY);
}
```

Figura 2.3.2: Código correspondiente al montaje.

Figura 2.3.3: Funcionamiento del sensor.

IV. Referencias

- https://www.electan.com/datasheets/flex22.pdf
- https://es.wikipedia.org/wiki/Divisor_de_tensi%C3%B3n
- https://www.luisllamas.es/medir-distancia-con-arduino-y-sensor-de-ultrasonidos-hc-sr04/
- https://github.com/Martinsos/arduino-lib-hc-sr04
- https://learn.adafruit.com/tmp36-temperature-sensor/using-a-temp-sensor

Práctica III

I. Funcionamiento de microservo

En este ejercicio se nos pedía controlar el movimiento del microservo utilizando un LDR.

Lo primero que hicimos fue montar el microservo como se muestra en el documento de los ejercicios, es decir, conectarlo a tierra, alimentación de 5V y a un pin digital (el número 13 en este caso) para poder indicarle qué posición debe tomar. Tras esto, comprobamos que, escribiendo los valores 0 y 90 sobre el servo, podíamos variar la posición de este, con un pequeño programa para depurarlo.

Una vez comprobado esto procedimos a montar un divisor de tensión como ya hemos hecho en ejercicios anteriores con un LDR y una resistencia de 330Ω , los cuales comprobamos antes con el multímetro y en el caso del LDR también poniendo y quitando el dedo para comprobar su correcto funcionamiento.
Cabe destacar que el LDR lo caracterizamos antes con el Serial Plotter, como ya hemos hecho anteriormente.

Tras terminar el proceso de montaje, quedó un layout como el que se aprecia en la Figura 3.1.1.

Se nos pedía que cuando el LDR detectase más oscuridad, el microservo girase 90 grados, y cuando el LDR detectase más luz, el microservo girase volviendo a su posición original.
Para hacer esto realizamos el código de la Figura 3.1.2, en el cual lo primero que hacemos es importar una biblioteca que nos permita trabajar con el microservo como un objeto, encapsulando el funcionamiento de este.

Tras ello, procedimos a inicializar el puerto serie, el objeto del servo y el pin de lectura del divisor de tensión, el cual declaramos como pin de entrada.

Posteriormente el código consiste en hacer la lectura del valor del LDR, comprobar el valor que obtenemos y, si está por debajo del valor umbral caracterizado anteriormente, consideramos que ha detectado oscuridad y que por tanto el microservo ha de abrirse (gira 90 grados).
En caso contrario, el microservo se cierra volviendo a su posición original (0 grados).
Todo esto se repetirá continuamente al encontrarse en *loop*.

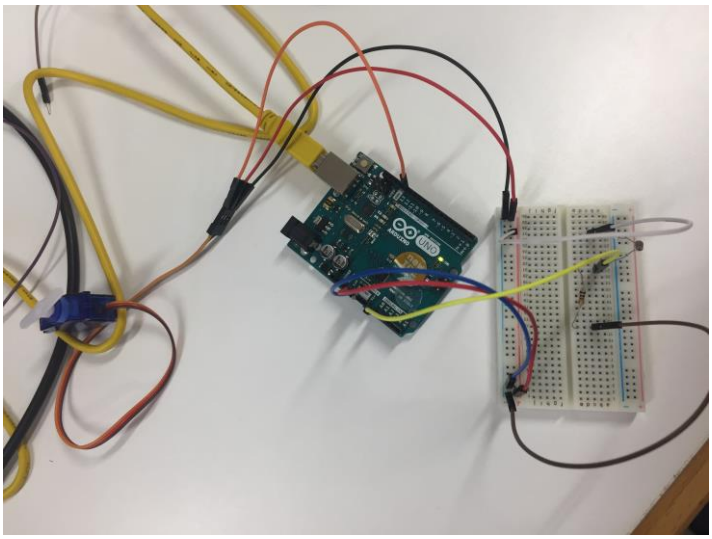

Figura 3.1.1: Montaje del circuito.

```
//Bibliotecas para controlar el servo
#include <Servo.h>
//Constantes:
//  + Pin de lectura del LDR
//  + Pin de control del servo
#define LDRPIN A0
#define SERVOPIN 13
//  + Valores de grados para abir y cerrar el servo
#define CLOSE 0
#define OPEN 90
//  + Valores minimos y maximos del LDR
#define MINVALUE 50
#define MAXVALUE 100
//  + Delay entre mediciones
#define DELAY 1000

Servo servo;

void setup() {
  Serial.begin(9600);
  servo.attach(SERVOPIN);
  pinMode(LDRPIN,INPUT);
}

void loop() {
  //leer datos del LDR
  int test = analogRead(LDRPIN);
  //Imprimirlos por pantalla
  Serial.println(test);
  //Mover el servo si hay poca luz y en caso contrario
  //   dejarlo en la posicion inicial
  if( test < MINVALUE )
    servo.write(OPEN);
  else
    servo.write(CLOSE);
  //Esperar delay segundos
  delay(DELAY);
}
```

Figura 3.1.2: Código correspondiente al montaje.

II. Funcionamiento de teclado de membrana

El montaje de este circuito lo tuvimos que hacer dos veces, una en el laboratorio (sin LCD), y otra en casa (con un LCD 1602), debido a que no teníamos cables suficientes para conectar el LCD (en gran parte porque fuimos poco previsores y no cogimos muchos cables al principio).

El primer montaje (Figura 3.2.1), que fue el realizado en el laboratorio fue muy sencillo, ya que solo tuvimos que conectar el teclado de membrana al Arduino, tras lo cual procedimos a programarlo.
El segundo montaje fue ampliación del anterior, ya que realizamos el mismo proceso de conectar el teclado de membrana al Arduino, tras lo cual realizamos las conexiones del LCD.
Este circuito no nos dio ningún problema, ya que el montaje era muy sencillo.

Tras terminar el primer montaje, elaboramos el código básico (el proyecto se llama tecladoMembrana), el cual no adjuntamos, ya que lo mejoramos para la segunda parte (el proyecto se llama keypadLCD). No obstante, es igual al código de la Figura 3.2.3, quitando la parte de declaración de constantes y manejador del LCD, y el "lcd.print" de la función "loop".
Volviendo a lo anterior, el código básico correspondiente al primer montaje consistía en declarar un objeto para gestionar el teclado (del que hablaremos más tarde), tras lo cual se inicializaba el puerto serie en la función "setup" y luego en la función loop, en la que cada 100 ms realizábamos un sondeo al teclado y, en el caso de haber un carácter, lo imprimíamos por pantalla.
En cuanto a el código del montaje en casa, consistió únicamente en tomar el código base y añadirle la declaración de las constantes y el manejador relativos al LCD, tras esto, en el setup inicializar el LCD y, por último, cuando imprimíamos el carácter por pantalla mediante el puerto serie, añadir una línea más para imprimirlo en el LCD.

En cuanto a los manejadores del teclado y el LCD, estos pertenecen a las bibliotecas de wrapper de estos.
El objeto LCD, quizás es el más sencillo de los 2, pues encapsula el enviar los códigos pertinentes al LCD para situar el cursor, limpiar la pantalla, enviar un carácter…
En cuanto a el objeto del keypad, es algo más complejo, porque este teclado usa el mecanismo que aprendimos en clase de teoría para conocer qué tecla ha sido pulsada, que consiste en que primero sondea las columnas y luego las filas. Y esto es lo que encapsula la biblioteca.

Por último, los resultados de estos experimentos no se pueden mostrar bien mediante imagen, pero se aprecian correctamente en el vídeo que se adjunta a los informes.
No obstante, en el LCD de la Figura 3.2.2, podemos observar que se han pulsado todos los caracteres, y que se han impreso por pantalla correctamente.

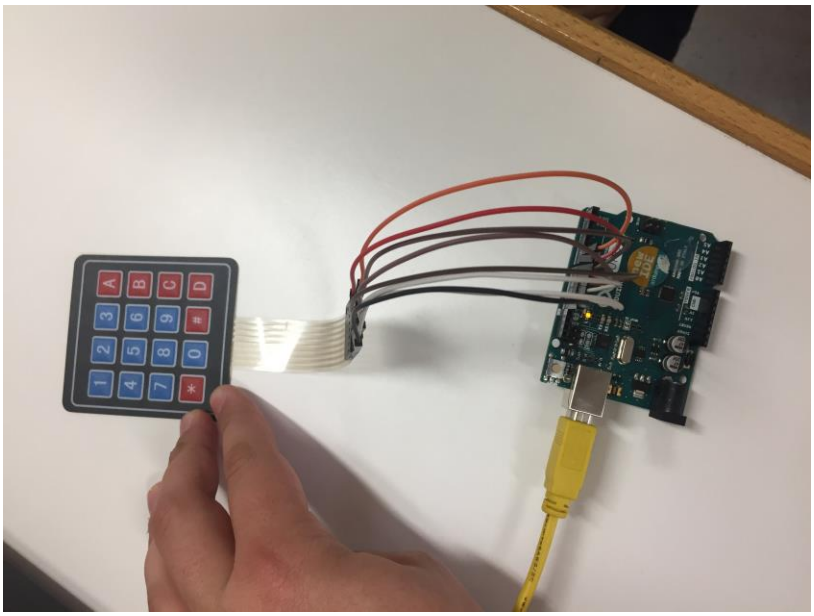

Figura 3.2.1 Montaje del circuito en el laboratorio

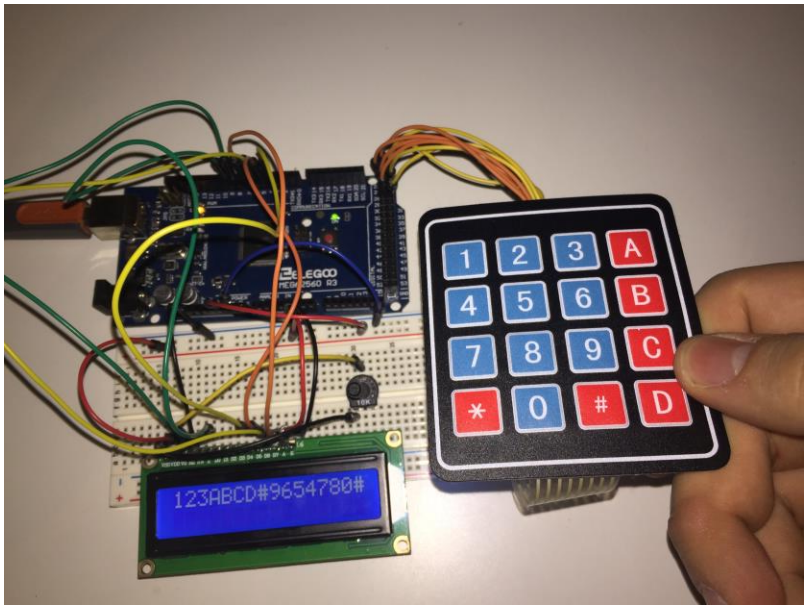

Figura 3.2.2 Montaje del circuito en casa.

```
//Bibliotecas para el teclado de membrana y el LCD respectivamente
#include <Keypad.h>
#include <LiquidCrystal.h>
//Constantes:
//  + Pines para conectar el LCD
//  + numero de filas y columnas del LCD
  #define RSPIN 8
  #define ENPIN 9
  #define DATA4 10
  #define DATA5 11
  #define DATA6 12
  #define DATA7 13
  #define NROW 2
  #define NCOL 16
//  + numero de filas y columnas del teclado
//  + pines en los que esta conectado el teclado
//  + valores de cada fila y columna del teclado
  const byte NROWS=4, NCOLS=4;
  byte colPins[NCOLS] = {26,27,28,29}, rowPins[NROWS] = {22,23,24,25 };
  char inputs[NROWS][NCOLS] = { {'1','2','3','A'}, {'4','5','6','B'}, {'7','8','9','C'}, {'*','0','#','D'} };
//  + delay de refresco de los LED
  #define DELAY 100
//Objeto manejador del teclado
  Keypad keypad = Keypad( makeKeymap(inputs),rowPins,colPins,NROWS,NCOLS);
//Objeto manejador del LCD
  LiquidCrystal lcd(RSPIN, ENPIN, DATA4, DATA5, DATA6, DATA7);

void setup() {
  // Inicializar el monitor serie
   Serial.begin(9600);
  // Inicializar el LCD con el número de  columnas y filas del LCD
  lcd.begin(NCOL, NROW);
}
void loop() {
  //Leemos un caracter
  char input = keypad.getKey();
  //en el caso de que haya un caracter
  if (input != 0){
    //Escribimos el caracer en la pantalla del PC
    Serial.println(input);
    //Escribimos en el LCD
    lcd.print(input);
  }
  delay(DELAY);
}
```

Figura 3.2.3 Código correspondiente a este montaje

III. Shield de sistema domótico:

Por último, tendremos que reconocer y estudiar los distintos componentes que tiene Shield (Figura 3.3.1), con la ayuda del esquema que vemos en la Figura 3.3.2. Algunos de estos componentes ya son conocidos: sensor de temperatura, LDR. Mientras que otros no, aunque no se nos hizo demasiado complicado.

El montaje de este circuito fue trivial, ya que consistió únicamente en poner el Shield sobre la placa de Arduino.

Para conseguir probar todos estos sensores, codificamos un programa que nos ayudo a hacer esto de forma sencilla. Ya que hay tres botones: menú, arriba y abajo; habilitamos en cada uno unos sensores diferentes.
Para ello lo que hicimos fue sondear los 3 botones, y en cuanto se pulsa un botón se entra a un modo hasta que se pulsa el mismo botón otra vez.
Los tres modos disponibles eran:

- **Botón menú (MENUBUTTON):**es el primer if dentro del do while, una vez hayamos pulsado el botón de menú, el programa empezará a leer los valores del LDR. Si este valor es menor que el que hemos elegido de umbral (caracterizado como en ejercicios anteriores)(MINIMUMLDR), sonará el BUZZER (activándolo con la función tone), que tras un delay, se apagará (con la función notone). Si volvemos a pulsar el botón de menú, volvemos al **loop** principal.
- **Botón arriba (UPBUTTON):** segundo if, ahora el programa mostrará por pantalla la lectura de temperatura del TMP36 (el proceso de medición es igual que el hecho en ejercicios anteriores).
- **Botón abajo (DOWNBUTTON):** tercer y último if. El programa leerá la temperatura y humedad usando el DHT, y los mostrará por pantalla. Si pulsamos otra vez el DOWNBUTTON, saldremos al loop principal.

De esta forma hemos comprobado el funcionamiento de los tres botones, para acceder a cada modo, y los diferentes sensores.

En cuanto a las dificultades, encontramos una única dificultad, que fue activar el sensor de humedad, ya que dejamos este Shield para el final, y al haber estado muchas personas en el laboratorio durante un par de horas, la humedad del ambiente subió y este no se estimulaba echándole el aliento.

Figura 3.3.1 Shield

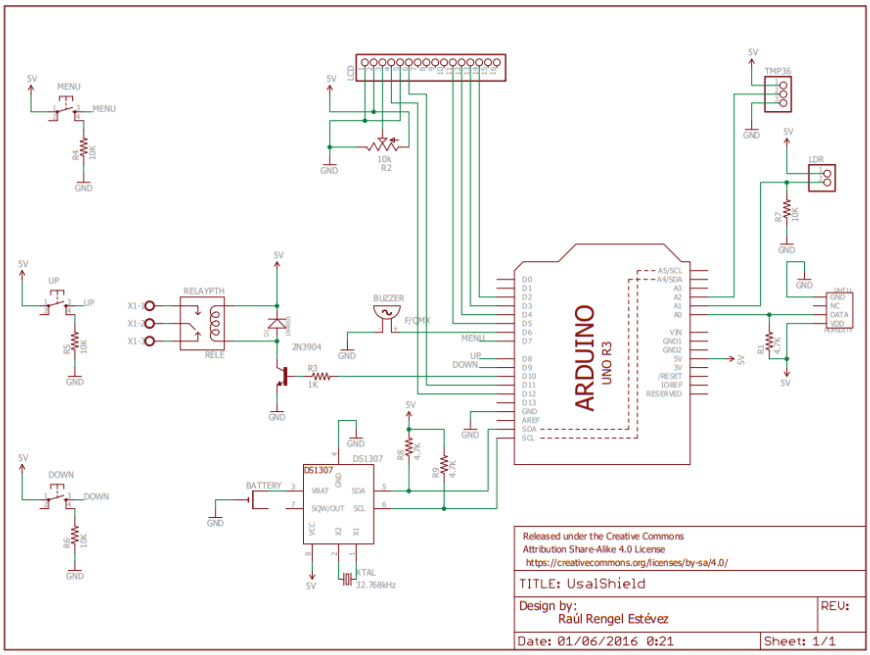

Figura 3.3.2 Esquema Shield

```
//Bibliotecas para controlar el sensor de temperatura y humedad
#include <DHT.h>
#include <DHT_U.h>
//Constantes:
//  + Pin para controlar el buzzer
//  + Frecuencia del buzzer
#define BUZZERPIN 6
#define BUZZERTONE 440
//  + Pin del boton de menu
//  + Pin del boton de up
//  + Pin del boton de down
#define MENUBUTTON 7
#define UPBUTTON 8
#define DOWNBUTTON 9
//  + Pin del LDR
//  + Valor minimo del LDR
#define LDRPIN A1
#define MINIMUNLDR 300
//  + Pin del sensor de temperatura
//  + Pin del sensor de temperatura y humedad DHT
#define TEMPERATURE A2
#define DHTPIN A0
//  + Delay entre mediciones
#define DELAY 1000

//Objeto manejador del sensor de temperatura y humedad
DHT dht(DHTPIN,DHT11);

void setup() {
  // Inicializar el monitor serie
  Serial.begin(9600);
  // Inicializar el pin del buffer
  pinMode(BUZZERPIN,OUTPUT);
  // Inicializar los pines de los botones
  pinMode(MENUBUTTON,INPUT);
  pinMode(UPBUTTON,INPUT);
  pinMode(DOWNBUTTON,INPUT);
  // Inicializar el pin del LDR
  pinMode(LDRPIN,INPUT);
  // Inicializar el pin del sensor de temperatura
  pinMode(TEMPERATURE,INPUT);
  // Inicializar el sensor DHT
  dht.begin();
}
```

Figura 3.3.3 Código: setup

```
void loop() {
    int data;

    //bucle infinito esperando a que se pulse en un boton
    do{
      //si se pulsa el boton de menu se entra en modo LDR y buzzer hasta que se vuelva a pulsar
      if(digitalRead(MENUBUTTON)){
        //imprimir el nuevo modo
        Serial.println("Menu Button pushed: entered in LDR and buzzer mode");
        //esperar para que no se confunda la medicion de condicion del while con el pulso del boton
        //   para entrar a este modo
        delay(DELAY);
        do{
            //si se le quita la luz a el LDR, hacemos sonar a el BUZZER por DELAY segundos
            if(analogRead(LDRPIN) < MINIMUNLDR){
                tone(BUZZERPIN,BUZZERTONE);
                delay(DELAY);
                noTone(BUZZERPIN);
            }
        //cuando se vuelva a pulsar el boton de menu, se sale del bucle y se vuelve a comenzar
        //   el loop
        }while(!digitalRead(MENUBUTTON));
        delay(DELAY);
        return;
      }else if(digitalRead(UPBUTTON)){
        //imprimir el nuevo modo
        Serial.println("Up Button pushed: entered in temperature measurement with TMP36");
        //esperar para que no se confunda la medicion de condicion del while con el pulso del boton
        //   para entrar a este modo
        delay(DELAY);
        do{
              //medir voltaje, convertirlo a celsius e imprimirlo por pantalla
              int medicion = analogRead(TEMPERATURE);
              float celsius = (medicion*5.0/1024 - 0.5)*100;
              Serial.print("Current temperature TMP36: "); Serial.println(celsius);
        //cuando se vuelva a pulsar el boton de up, se sale del bucle y se vuelve a comenzar
        //   el loop
        }while(!digitalRead(UPBUTTON));
        delay(DELAY);
        return;
      }else if(digitalRead(DOWNBUTTON)){
        Serial.println("Down Button pushed: entered in temperature and humdity measurement with DHT");
        //esperar para que no se confunda la medicion de condicion del while con el pulso del boton
        //   para entrar a este modo
        delay(DELAY);
        do{
              //Medir temperatura e imprimirla por pantalla
              Serial.print("Current temperature DHT: "); Serial.print(dht.readTemperature());
              //Medir humedad e imprimirla por pantalla
              Serial.print("   Current humidity DHT:"); Serial.println(dht.readHumidity());
        //cuando se vuelva a pulsar el boton de down, se sale del bucle y se vuelve a comenzar
        //   el loop
        }while(!digitalRead(DOWNBUTTON));
        delay(DELAY);
        return;
      }
    }while(1);
```

Figura 3.3.4 Código: loop

IV. Referencias

- https://www.naylampmechatronics.com/blog/34_Tutorial-LCD-conectando-tu-arduino-a-un-LCD1.html
- https://www.intorobotics.com/tutorial-how-to-control-the-tower-pro-sg90-servo-with-arduino-uno/
- https://www.prometec.net/buzzers/
- https://www.arduino.cc/reference/en/language/functions/advanced-io/tone/
- https://www.arduino.cc/reference/en/language/functions/advanced-io/notone/