



LABORATORIO DE DESARROLLO HARDWARE - MEMORIA DE ARDUINO



José Manuel Jarana Expósito

Objetivos.....	2
Introducción.....	2
<i>Arduino.....</i>	<i>2</i>
<i>Relé.....</i>	<i>3</i>
<i>Pull-Up y Pull-Down.....</i>	<i>3</i>
<i>Motor de DC.....</i>	<i>4</i>
<i>Servo.....</i>	<i>4</i>
<i>LCD.....</i>	<i>4</i>
<i>Sensor de temperatura.....</i>	<i>5</i>
<i>Display de 7-segmentos.....</i>	<i>5</i>
Desarrollo de la práctica	5
1. <i>Controlar el encendido y apagado de un led mediante el puerto serie y un pequeño programa en Python (Mejora: Controlar el apagado y encendido de una bombilla con un relé, con el mismo programa de Python).</i>	<i>5</i>
2. <i>Controlar el encendido y apagado de un led con un pulsador (Pull-Up o Pull-Down) mediante interrupciones en Arduino.</i>	<i>7</i>
3. <i>Control del ángulo de posicionamiento del servo desde el pc, enviando el valor a través del puerto serie con la ayuda de un programa en Python.</i>	<i>8</i>
4. <i>Control de velocidad de giro de un motor de continua en doble sentido con un potenciómetro. .</i>	<i>10</i>
5. <i>Impresión en una pantalla LCD. Primero imprimimos nuestro nombre y el nombre de la asignatura. Después imprimiremos en la pantalla información que le enviamos por el puerto serie. ..</i>	<i>12</i>
6. <i>Empleando el sensor de temperatura tmp36GZ, diseñaremos un termómetro que nos dé la temperatura en tiempo real.</i>	<i>16</i>
7. <i>Contador de 00 a 59 en el display de 7-segmentos cada segundo.</i>	<i>17</i>
Conclusiones	21

Objetivos

Nuestro objetivo en las prácticas que daremos sobre Arduino será en un principio conocer mejor esta plataforma de desarrollo de hardware abierta y las prestaciones de las que dispone, así como sus modos de programación. Al principio lo que haremos será probar distintos ejemplos que ya viene incluidos en el software de la propia placa. Para ello utilizaremos componentes de esta que viene incluidos en el starter kit, tales como leds, relés, transistores, regletas para hacer el cableado correcto de los componentes, etc. Además de otros que también nos proporcionará como bombillas y demás. Los distintos objetivos que seguiremos durante las prácticas son:

1. Controlar el encendido y apagado de un led mediante el puerto serie y un pequeño programa en Python (Mejora: Controlar el apagado y encendido de una bombilla con un relé, con el mismo programa de Python). PD: Practica 1
2. Controlar el encendido y apagado de un led con un pulsador (Pull-Up o Pull-Down) mediante interrupciones en Arduino.
3. Control del ángulo de posicionamiento del servo desde el pc, enviando el valor a través del puerto serie con la ayuda de un programa en Python.
4. Control de velocidad de giro de un motor de continua en doble sentido con un potenciómetro. PD: practica 2.
5. Impresión en una pantalla LCD. Primero imprimimos nuestro nombre y el nombre de la asignatura. Después imprimiremos en la pantalla información que le enviamos por el puerto serie.
6. Empleando el sensor de temperatura tmp36GZ, diseñaremos un termómetro que nos dé la temperatura en tiempo real.
7. Contador de 00 a 59 en el display de 7-segmentos cada segundo.

Introducción

Arduino

Lo primero que vamos a hacer es definir qué es Arduino. Arduino es una plataforma de desarrollo de hardware abierta fácil de usar tanto a nivel software como en Hardware con una gran flexibilidad ya que es de código abierto.

La capacidad que tiene Arduino de tomar información de su entorno es una de las grandes ventajas que proporciona esta placa añadiendo módulos para que tenga sensores, conexión WI-FI y demás características.

Debido a esta versatilidad y a su bajo coste, se ha creado una gran comunidad detrás de este entorno que nos permite ver y ejecutar códigos y librerías desde las más sencillas para tareas comunes o estudiantiles, hasta algunas con un carácter más profesional.

Existen varias versiones de la placa, que varían según la necesidad que nosotros tengamos. La más básica y la que nosotros vamos a usar es la versión de microcontroladores ARM de 8-bits (Arduino/Genuino Uno) aunque existen versiones de hasta 32-bits o que implementa ya de por sí la placa el módulo de conexión WI-FI. La nuestra, como dijimos antes, implementa un microcontrolador de 8-bit, con 14 I/O digitales (6 de las cuales pueden usarse como salidas PWM), 6 analógicas, 32KB de memoria flash, 2 KB de RAM. También trabaja a una frecuencia de 16 MHz y con un voltaje de 5 V.

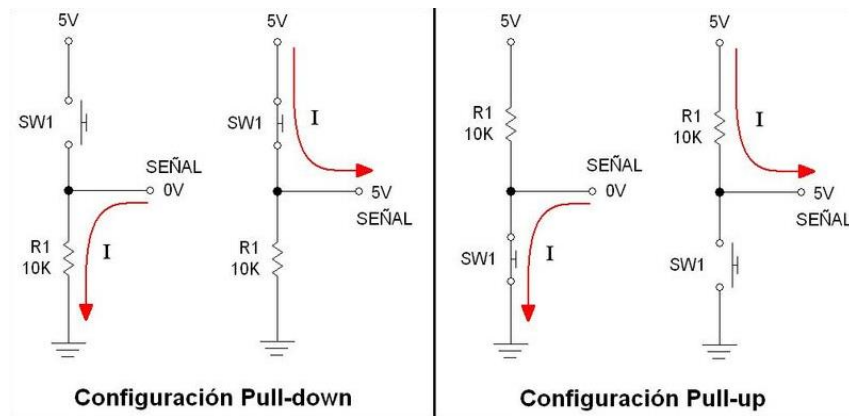
El IDE para desarrollar que nos aporta es totalmente gratuito, y se puede descargar de la página oficial para cualquier sistema operativo (<https://www.arduino.cc/en/Main/Software>). Este entorno tiene un uso sencillo y nos permite subir a la placa el programa que desarrollaremos. El lenguaje de programación que utiliza es uno propio que esté basado en Processing, que es a su vez muy parecido a C++.

Relé

Este dispositivo electromagnético funciona como un interruptor controlador por un circuito eléctrico. Por medio de una bobina y un electroimán, se acciona un juego de uno o varios contactos que permiten abrir o cerrar otros circuitos eléctricos independientes. En nuestro caso, por defecto está en contacto con una salida y si le metemos una tensión "HIGH" conmuta a la otra, ya que activa el imán interior y acciona la armadura para que cambie.

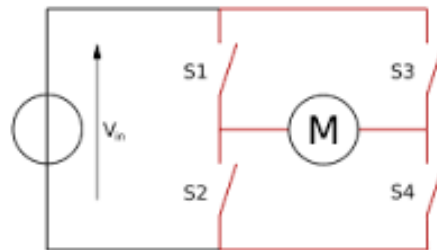
Pull-Up y Pull-Down

A la hora de la conexión de componentes electrónicos para evitar interferencias o ruidos y que los valores que den 1 lógico o 0 lógico, utilizaremos las resistencias de pull-Up y pull-Down. En la resistencia de pull-Down conectamos una resistencia a tierra y a su vez eso también lo conectamos a la salida, por lo tanto, sin pulsar el botón nos dará una salida de "0". Cuando pulsemos el botón, la señal saldrá por nuestra salida, ya que por la rama de tierra tiene una resistencia. Por el contrario, la resistencia de Pull-Up conectamos la resistencia en el camino de la señal de entrada a la señal de salida por lo que continuamente da una salida de "1", y pulsador estará conectado en un camino entre la señal de salida y tierra, para que cuando se pulse cree camino directo a GND generando un "0". La propia placa podemos ponerle a la hora de programar que la salida sea por Pull-Up o pull-Down, ya que la propia placa trae una resistencia interna para que pueda implementar esta conexión. Vemos un ejemplo esquemático:



Motor de DC

Este motor funciona gracias a que convierte la energía eléctrica que le llega en un movimiento rotatorio debido a la acción de un campo magnético ya que el conductor que ejecuta el giro sufre una fuerza magnética producida por este campo. El motor puede girar en ambos sentidos, dependiendo de la polaridad de la señal que le llegue. Según sea la polaridad que le llegue variará también la polaridad de los imanes que crean el campo magnético en el interior del motor. Esta polaridad la iremos cambiando gracias a la implementación por Hardware de un puente H como el de la figura. En este caso si accionamos los interruptores S1 y S4, el motor girará en un sentido, por el contrario, si activamos S2 y S3 girará en el sentido opuesto.



Servo

El servo es un dispositivo actuador que consiste básicamente en un motor de DC y un dispositivo de control que nos permitirá girar para colocarse en un ángulo determinado que va desde 0° de 180°. El ángulo del servo lo modificaremos dependiendo de la señal que le enviaremos, que en nuestro caso será mediante una señal de PWM que envía uno de los pines que permite esto en Arduino. Dependiendo del ancho del pulso que le enviemos así se modificará el ángulo, a mayor ancho de pulso, mayor ángulo.

LCD

Los displays LCD están compuestos por un "cristal líquido" retenido entre dos placas transparentes conductoras. Dicho cristal cambia su estado de visualización dependiendo del potencial a que se somete las placas conductoras (nosotros podremos modificar este contraste con un potenciómetro) estas deben estar divididas en pequeñas celdas controlables independientemente para poder activar cada elemento que compone el visualizador. El display

consta de dos líneas y para elegir en la línea que queremos escribir utilizaremos un comando en Arduino que nos permite poner el curso donde queramos.

Sensor de temperatura

También llamado termistor, basa su funcionamiento en la variación de la resistividad que da un semiconductor con la temperatura. El termistor tmp36gz que es el que nosotros hemos utilizado, consta de tres patillas que conectaremos a VCC y GND los extremos, y la patilla del medio nos proporcionará una señal analógica que será la lectura de la temperatura en forma de voltaje, en un rango de medición en este sensor de -50°C a 125°C. Para hacer esta conversión debemos utilizar la fórmula:

$$\text{Temperatura } ^\circ\text{C} = 100 * (\text{Lectura de Voltaje}) - 50$$

Display de 7-segmentos

Shield de Arduino que viene con el display de 7 segmentos. Para utilizarlo correctamente debemos añadir las funciones que convierte el valor que entra a una correcta visualización del 7 segmentos. Estas funciones se pueden encontrar en la página de Arduino mismo, y las añadimos al código al final. Debemos tener en cuenta también, que para que nuestros ojos puedan percibir el número que se muestra, debemos estar haciendo un continuo refresco para que los leds del display puedan verse correctamente.

Desarrollo de la práctica

Aquí pondremos los diferentes códigos y pautas que hemos realizado para cumplir los objetivos que nos propusimos hacer para la plataforma de Arduino.

- 1. Controlar el encendido y apagado de un led mediante el puerto serie y un pequeño programa en Python (Mejora: Controlar el apagado y encendido de una bombilla con un relé, con el mismo programa de Python).*

A la hora de montar la placa en este caso vamos a controlar el pin 13 que lo conectaremos a directamente a un led, o si queremos podemos mirar el propio led del pin 13 que ya trae la placa de Arduino.

El programa será para que acceda al puerto serie a una frecuencia por defecto (en nuestro caso 9600 baudios) para que la placa y el puerto serie estén a la misma y se puedan entender. Además de poner el led 13 como salida y activarlo en alto en caso de que por el puerto serie le enviemos una "H" o lo pondríamos en apagado cuando se envíe "L". Este sería nuestro código:

```

int led = 13;
void setup () {
    pinMode(led, OUTPUT); //LED 13 como salida
    Serial.begin(9600); //Inicializo el puerto serial a 9600 baudios
}

void loop () {
    if (Serial.available()) { //Si está disponible
        char c = Serial.read(); //Guardamos la lectura en una variable char
        if (c == 'H') { //Si es una 'H', enciendo el LED
            digitalWrite(led, HIGH);
        } else if (c == 'L') { //Si es una 'L', apago el LED
            digitalWrite(led, LOW);
        }
    }
}

```

Como ya dijimos esto lo subimos a la placa, y ahora haremos el programa en Python que nos permita enviarle los comandos necesarios a Arduino:

```

import serial

arduino = serial.Serial('/dev/ttyACM0', 9600)

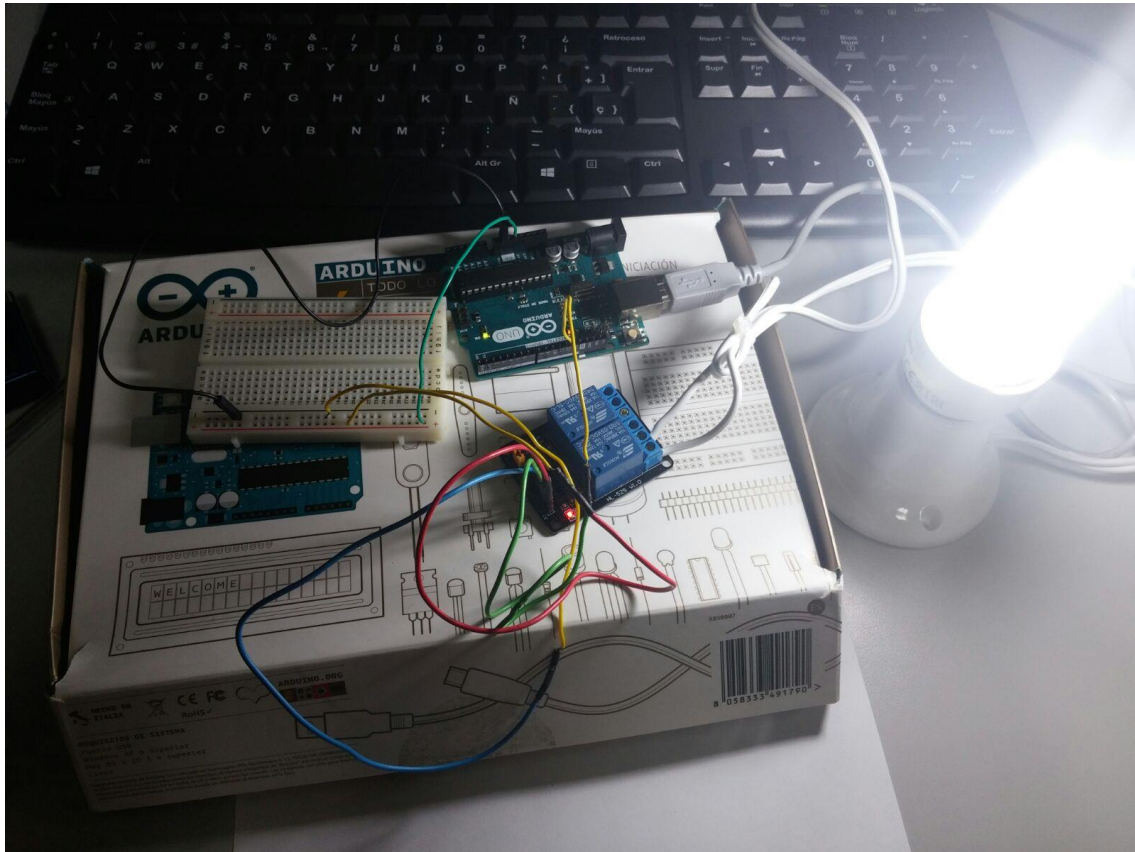
print("Starting!")

while True:
    comando = raw_input('Introduce un comando: ') #Input
    arduino.write(comando) #Mandar un comando hacia Arduino
    if comando == 'H':
        print('LED ENCENDIDO')
    elif comando == 'L':
        print('LED APAGADO')

arduino.close() #Finalizamos la comunicacion

```

Tras esto, en vez de encender un led, lo que haremos será conectar un relé, al que se le enviará la señal de "HIGH" o "LOW", y conmutará entre una u otra conexión. Después de esto, conectaremos la bombilla a una conexión del relé y la conectaremos a un voltaje de 220V. Ahora según la señal que le enviemos conmutará el relé y se encenderá o no, como vemos en la foto:



2. Controlar el encendido y apagado de un led con un pulsador (Pull-Up o Pull-Down) mediante interrupciones en Arduino.

Las conexiones que haremos en la placa será el interruptor con el pin 2, que será el pin de interrupción por Pull-Up (ya implementado en la placa y definida en el código así). Cuando pulsemos el botón una vez se encenderá el led, aunque podemos cambiar este modo cambiando una palabra en el código, como por ejemplo para que solo esté encendida mientras está pulsado. El código en Arduino:

```
const byte ledPin = 13;
```



```

const byte interruptPin = 2;
volatile byte state = LOW;

void blink();

void setup() {
    pinMode(ledPin, OUTPUT);
    pinMode(interruptPin, INPUT);
    attachInterrupt(digitalPinToInterrupt(interruptPin), blink, FALLING);
    //Si cambiamos FALLING por otro modo, cambio el modo de encenderse
}

void loop() {
    digitalWrite(ledPin, state);
}

void blink () {
    state =! state;
}

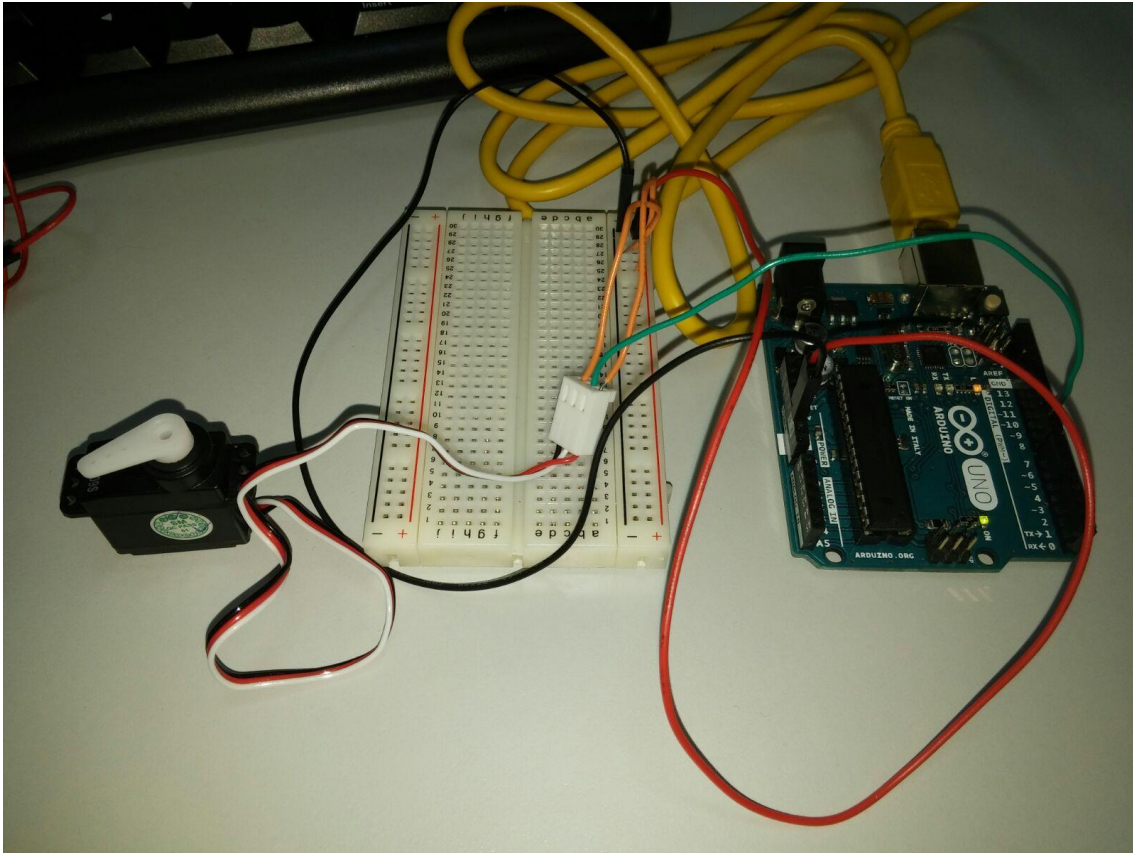
```

Este punto nos produce a veces un pequeño error respecto al pulsador el cual no responde a veces bien al pulsarlo, y a lo mejor puede que de un pequeño parpadeo y vuelva al estado que estaba anteriormente de pulsarlo. Esto se debe a que el pulsador, cuando se pulsa, puede producir rebotes y por lo tanto dar un segundo valor erróneo. Una solución propuesta para esto es que en vez de leer directamente del pulsador el pin que provoca la interrupción, lo que hacemos es que el pulsador lo conectamos a otro pin de la placa, después de leer el valor que viene del pulsador, le metemos un delay, y entonces ese valor recogido en otro pin que utilizamos de auxiliar es el que introducimos como valor al pin de interrupción. Gracias al delay que le metemos en el pin de antes, evita que pueda coger valores de rebote después de coger el primero.

3. Control del ángulo de posicionamiento del servo desde el pc, enviando el valor a través del puerto serie con la ayuda de un programa en Python.

Al igual que en el primer punto de la práctica, debemos enviarle por comando del pc el ángulo al que queremos que se posicione el servo gracias a un programa en Python. Procederemos al

montaje de los componentes, el cual es muy simple, lo único que debemos hacer es conectar el servo a corriente y la conexión que lo maneja al pin que utilizaremos en la placa como vemos en la fotografía.



Lo primero será hacer el pequeño programa en Python, que nos permita introducir el ángulo por el terminal a la placa.

```
Import serial

Arduino=serial.Serial('/dev/ttyACM1', 9600)
print ("Starting!")
while True:
    comando = raw_input('Introduce el angulo: ') #Input
    arduino.write(comando) #Mandar un comando hacia

arduino.close()
```

El código que subiremos a la placa es el siguiente:

```
#include <Servo.h>
```

```

Servo myservo;
int val;

void setup () {
    Serial.begin(9600); //Inicializo el puerto serial a 9600 baudios
    myservo.attach(9);
}

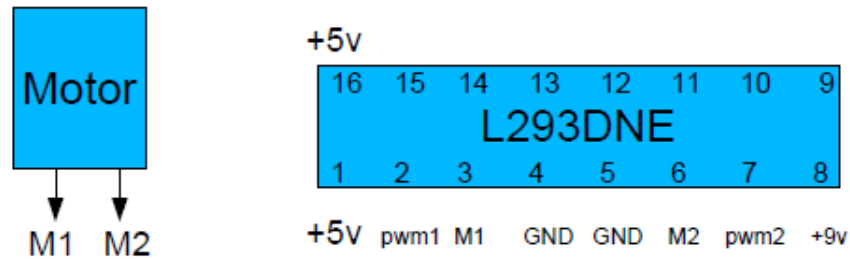
void loop () {
    if (Serial.available()) {           //Si está disponible
        val = Serial.parseInt();        //Guardamos la lectura del angulo
        if(val>=0 && val<=180){
            myservo.write(val);          //Escribimos el valor en el servo
        }
    }
}

```

Como vemos, hemos cambiado la función `serial.read()` por `Serial.parseInt()`, para que así nos permita coger el valor numérico directamente. Sin embargo, esta función tiene un pequeño inconveniente, y es el gran retraso que lleva esta función, siendo aproximadamente de 1 segundo desde el envío de datos hasta el movimiento en el ángulo que le enviamos. Si queremos acelerar esto, debemos utilizar de nuevo la función `Serial.Read()`, y así pasamos carácter a carácter el valor que enviamos y a cada carácter lo pasamos a int. Aunque el código sea más largo o parezca más liso, se nota mucho en la velocidad de reacción del servo.

4. Control de velocidad de giro de un motor de continua en doble sentido con un potenciómetro.

En este apartado lo que haremos será controlar la velocidad de giro de un motor en doble sentido con un potenciómetro. Para hacer esto, al poner el potenciómetro en el punto medio, el motor se parará, mientras que, si lo llevamos a uno de los extremos, girará a máxima velocidad en un sentido u otro. Para hacer esto contaremos con el datachip L293DNE que nos permite controlar el motor con la señal PWM y alimentarlo con los 9V que necesita. La conexión la haremos de la siguiente forma:



Ahora crearemos el código en Arduino, que nos permita manejar ambos sentidos en el motor. El principal problema que tenemos es que la entrada analógica que da el potenciómetro es de 10 bits(valor de 0 a 1023), sin embargo, a la hora de meterle la señal de PWM necesitamos meterle un valor de 8 bits(valor entre 0 y 255). Además, esto debemos dividirlo en dos ya que debemos de cambiar el sentido. Por tanto, si el potenciómetro nos da un valor entre 0 y 512, girará en un sentido, y si lo hace entre 512 y 1023 lo hará en el otro.

```
int pin2=9;    //Entrada 2 del L293D
int pin7=10;   //Entrada 7 del L293D
int pote=A0;   //Potenciómetro

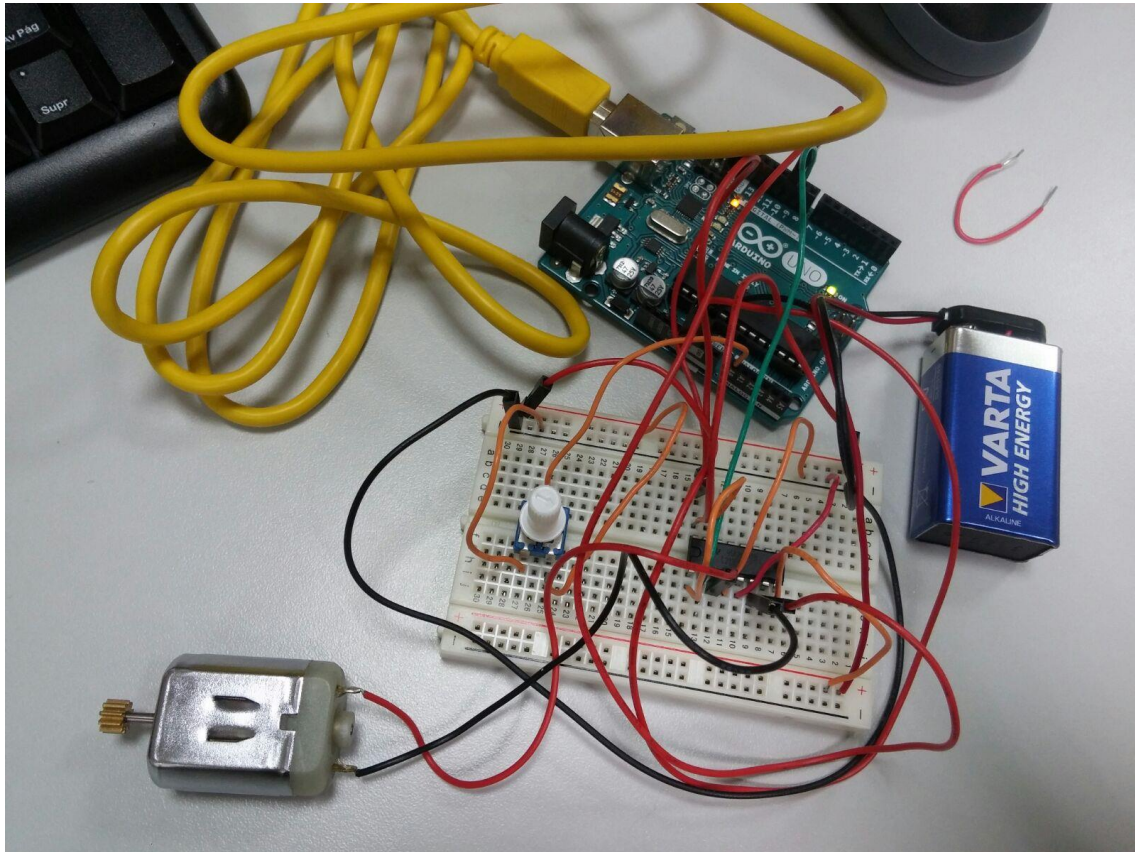
int valorpote;    //Variable que recoge el valor del potenciómetro
int pwm1;         //Variable del PWM 1
int pwm2;         //Variable del PWM 2

void setup()
{
    //Inicializamos los pins de salida
    pinMode(pin2,OUTPUT);
    pinMode(pin7, OUTPUT);
}

void loop()
{
    //Almacenamos el valor del potenciómetro en la variable
    valorpote=analogRead(pote);

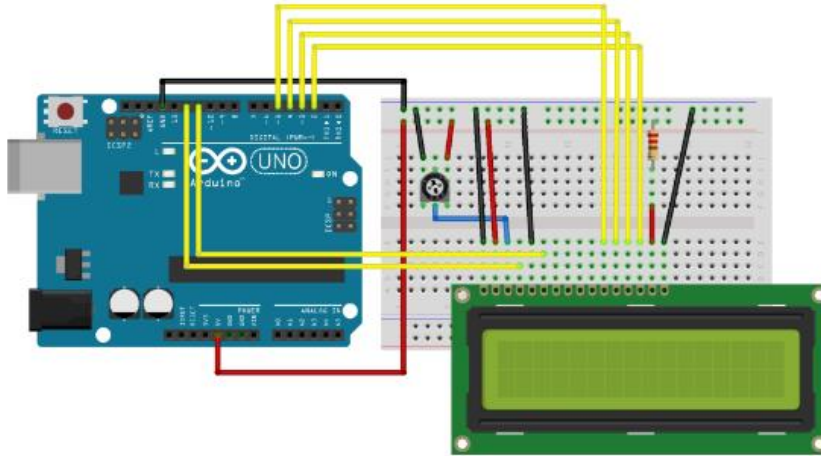
    //Hacemos el mapeado de la salida del potenciómetro y de la entrada de arduino
    pwm1 = map(valorpote, 0, 1023, 0, 255);
    pwm2 = map(valorpote, 0, 1023, 255, 0); //El PWM 2 esta invertido respecto al PWM 1
```

```
//Sacamos el PWM de las dos salidas usando analogWrite(pin,valor)
analogWrite(pin2,pwm1);
analogWrite(pin7,pwm2);
}
```



5. *Impresión en una pantalla LCD. Primero imprimimos nuestro nombre y el nombre de la asignatura. Después imprimiremos en la pantalla información que le enviamos por el puerto serie.*

Haremos una pequeña prueba con la pantalla LCD en Arduino, en la que pondremos nuestros datos solamente. Montaremos la pantalla LCD tal y como se indica en la imagen.



Como podemos ver, añadimos además un potenciómetro, que lo que hará será modificar el contraste de la pantalla LCD.

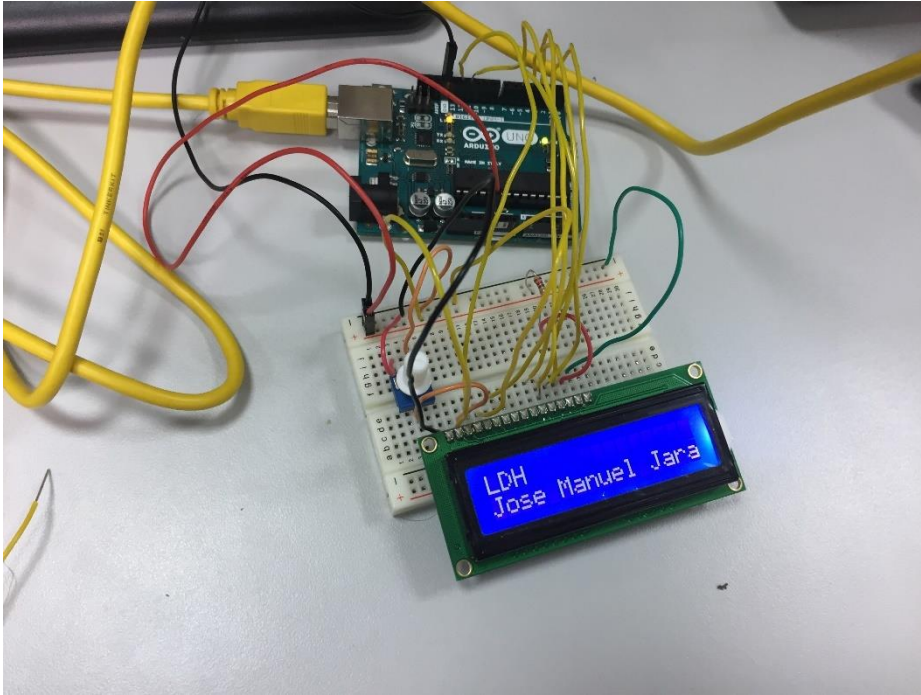
El primer código que subimos a la placa para que nos muestre estos datos es este:

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  // Inicializa el LCD en columnas y filas
  lcd.begin(16, 2);
  // Imprime el mensaje
  lcd.print("LDH");
}

void loop() {
  //Pone el cursor en la columna 0, línea 1
  lcd.setCursor(0, 1);
  lcd.print("Jose Manuel Jarana");
}
```

Aquí ahora veremos una pequeña prueba de nuestra impresión en la pantalla LCD, junto con el montaje que hicimos para que nos la muestre.



Después de esta pequeña prueba y comprobar que la pantalla LCD y la conexión está hecha de la manera correcta, ahora procederemos a enviarle unos datos desde el Pc por el puerto serie. Como en otras ocasiones, crearemos primero nuestro programa en Python, para el puerto serie.

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  
  
import serial  
  
arduino = serial.Serial('/dev/ttyACM0', 9600)  
  
while True:  
    comando = raw_input('Introduce cadena: ') #Input  
    arduino.write(comando) #Mandar un comando hacia Arduino  
  
arduino.close() #Finalizamos la comunicacion
```

El código de Arduino que esta vez subiremos será este:

```
#include <LiquidCrystal.h>  
  
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  
  
void setup() {
```

```

// Inicializa el LCD en columnas y filas
lcd.begin(16, 2);
Serial.begin(9600);
}

void loop() {
  // when characters arrive over the serial port...
  if (Serial.available()) {

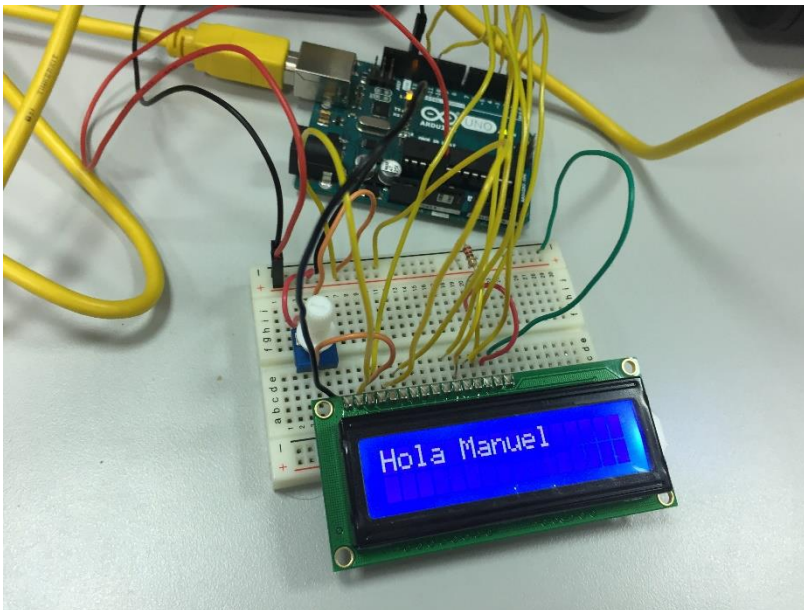
    delay(100);
    // Limpiamos la pantalla
    lcd.clear();
    // Lee todos los caracteres
    while (Serial.available() > 0) {

      lcd.write(Serial.read());

    }
  }
}

```

El resultado que veremos esta vez es este, tras enviar por el PC "Hola Manuel".



6. *Empleando el sensor de temperatura tmp36GZ, diseñaremos un termómetro que nos dé la temperatura en tiempo real.*

Conectaremos el sensor de temperatura al pin 0 de nuestra placa, para que reciba los datos que nos envía. Como dijimos en la introducción, este sensor dar un voltaje según la temperatura, que tendremos que convertir para que nos muestre el voltaje. Esta conversión la veremos en el código mostrado a continuación.

```
#include <LiquidCrystal.h>

// Inicialica los pines de la pantalla LCD
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
int sensorPin=0;

void setup() {
  // Inicializa el LCD en columnas y filas
  lcd.begin(16, 2);
  Serial.begin(9600);
}

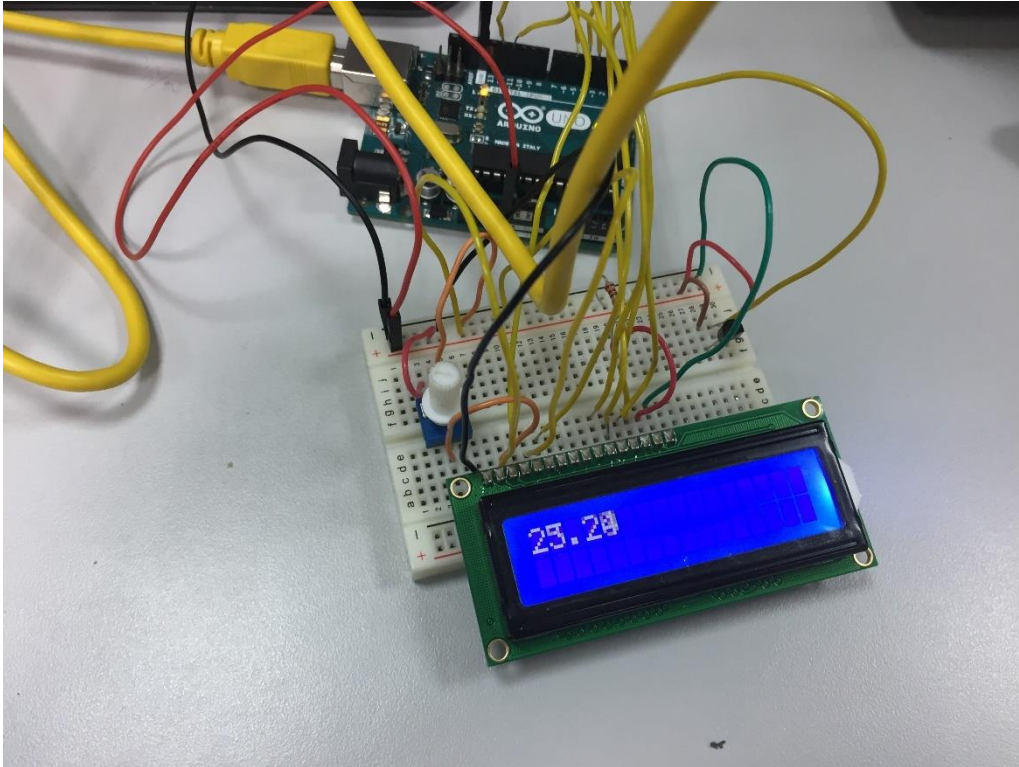
void loop() {
  // Espera el carácter le llegue
  delay(100);
  // clear the screen
  lcd.clear();

  //Lee el dato que nos envía
  int reading = analogRead(sensorPin);

  // Convertimos la lectura en voltaje
  float voltage = reading * 5.0;
  voltage /= 1024.0;

  // Hacemos la conversión de voltaje a grados
  float temperatureC = (voltage - 0.5) * 100 ;
```

```
// Lo imprimimos por pantalla  
lcd.print(temperatureC)  
}
```



Al hacerlo así tenemos un pequeño problema, y es que el LCD fluctúa muchísimo, y a veces apenas se puede ver bien los dígitos de lo rápido que va cambiando. En vez de eso, podemos almacenar unos 10 valores en un tiempo, y después mostrar la media aritmética de los diez valores, lo que nos daría más tiempo a verlo correctamente y sin embargo no modificaríamos apenas el resultado final, ya que en ese tiempo mínimo apenas varía la temperatura.

7. Contador de 00 a 59 en el display de 7-segmentos cada segundo.

Le vamos a conectar un Shield que tiene un contador 7-segmentos. Ahora como veremos en el código, tenemos que hacer la conversión de los valores del contador a 7-segmentos para mostrarlo. También, para que el contador sea visible, lo que tenemos que hacer es refrescar varias veces el dato mostrado.

```

int segPins[] = {
  0, 1, 2, 3, 4, 5, 6, 7};

int tiempototal= 1000;
int j = 40;

int disp1 =8;
int disp2= 9;

int dat1 = 0;
int dat0 = 0;

int cont = 0; //Variable Auxiliar que utilizamos de contador

void setup() {
  // put your setup code here, to run once:
  // loop over the pin array and set them all to output:
  for (int thisseg = 0; thisseg < 8; thisseg++) {
    pinMode(segPins[thisseg], OUTPUT);
  }
  pinMode(disp1, OUTPUT);
  pinMode(disp2, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  //  dat1 = 0;
  //  dat0 = 0;

  //Llamada a función refresh pasando los datos correctos
  dat0=cont / 10;
  dat1=cont % 10;
  refresh(dat1,dat0);
  if(cont<59){
    cont=cont + 1;
  }
}

```

```

    else{
        cont=0;
    }
// Cálculo correcto de los datos dat1, dat0 --> desde 0 0 hasta 5 9
}

// Función refresh: Duración total de ejecución de refresh: tiempototal.
//Se van intercambiando los displays (disp1 con dat0 y disp2 con dat1) a una
frecuencia que evite el parpadeo (j--> numero de veces que se activan ambos d
isplays)
void refresh( int data1, int data0) {
    int tiempo_refresco = tiempototal/(2*j);

// Bucle de activación de los displays. El bucle se ejecuta j veces. Para esc
ribir en los displays se llama a la función write_data (dato)
    for (int i=0; i<=j; i++){
        digitalWrite(disp1, 0);
        digitalWrite(disp2, 1);
        write_data(dat1);
        delay(tiempo_refresco);
        digitalWrite(disp1, 1);
        digitalWrite(disp2, 0);
        write_data(dat0);
        delay(tiempo_refresco);
    }
}

// Función write_data(dato): Transforma dato en su código siete segmentos par
a escribirlo en el display

void write_data (int arg) {
    switch (arg) {
        case 0:
            //do something when var equals 1
            write7seg(0x7e);
            break;
        case 1:
            //do something when var equals 2

```

```

        write7seg(0x30);
        break;
case 2:
    //do something when var equals 1
    write7seg(0x6d);
    break;
case 3:
    //do something when var equals 2
    write7seg(0x79);
    break;
case 4:
    //do something when var equals 1
    write7seg(0x33);
    break;
case 5:
    //do something when var equals 2
    write7seg(0x5b);
    break;
case 6:
    //do something when var equals 1
    write7seg(0x1f);
    break;
case 7:
    //do something when var equals 2
    write7seg(0x70);
    break;
case 8:
    //do something when var equals 1
    write7seg(0x7f);
    break;
case 9:
    //do something when var equals 1
    write7seg(0x73);
    break;
    }
}

```

```
// Función write7seg(dato_7seg): escribe el valor dato_7seg en el display

void write7seg (unsigned char arg) {
    unsigned char segmen = 0x01;
    unsigned char display1;
    display1 = arg;

    for (int i = 0; i < 8; i++) {
        if ((display1 & segmen) == 0x00)
            digitalWrite(i, LOW);
        else
            digitalWrite(i, HIGH);

        segmen <<= 1; }
}
```

Conclusiones

Tras todas estas prácticas de Arduino, hemos podido comprobar que es una placa muy versátil y gran parte de esa versatilidad es a la gran comunidad que también tiene por detrás que dan códigos y bibliotecas para utilizar la placa de distintas formas y diversos usos. Además, también hay bastantes periféricos y shield compatibles. Uno de los usos comunes de esta placa es la utilizar los sensores y la domotización, como por ejemplo en la práctica de medir la temperatura, haciendo que salte el aire acondicionado si detecta una temperatura determinada.

Además de los componentes y la comunidad que tiene detrás, también ayuda que sea un lenguaje sencillo de entender y comprender para cualquier persona que tenga una mínima idea de programación.

