

Plataforma Arduino

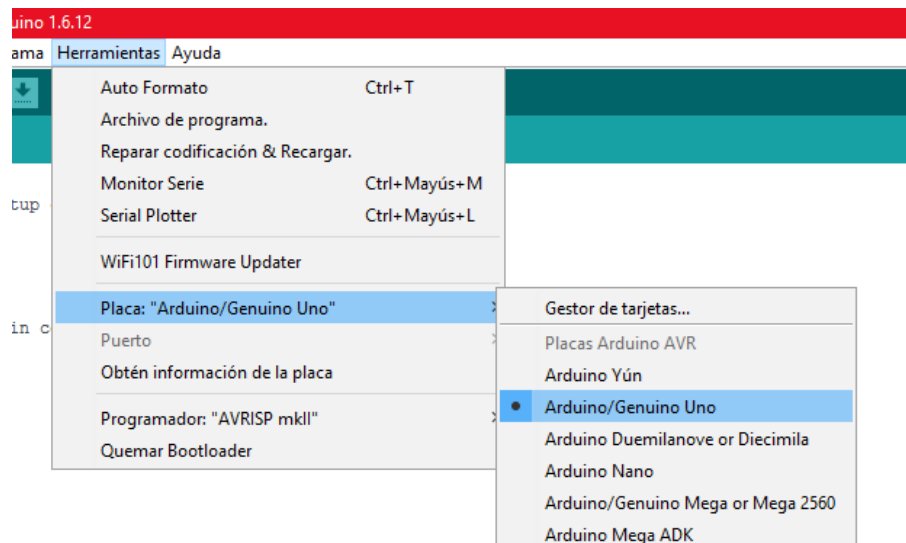


Antonio Manuel Núñez Domínguez

Sesión 1 de laboratorio

En esta primera sesión se ha empezado siguiendo un tutorial de un proyecto de internet. Seguidamente se han ido incorporando nuevas formas de utilizar el mismo proyecto, pero con distintos componentes, tales como un relé y una bombilla.

El proyecto consiste en controlar el encendido y apagado de un **LED**. Para ello, lo primero que se ha hecho es instalar el entorno de **Arduino**, y configurarlo, seleccionando la placa **Arduino** que se va a utilizar y el puerto al que se va a conectar. En este caso, **Arduino UNO**.

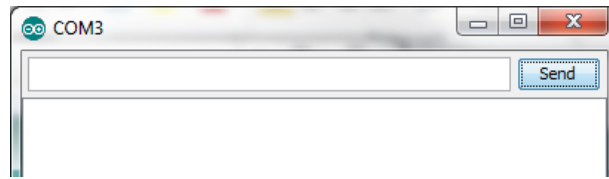


Se introduce el código del programa en el entorno **Arduino**.

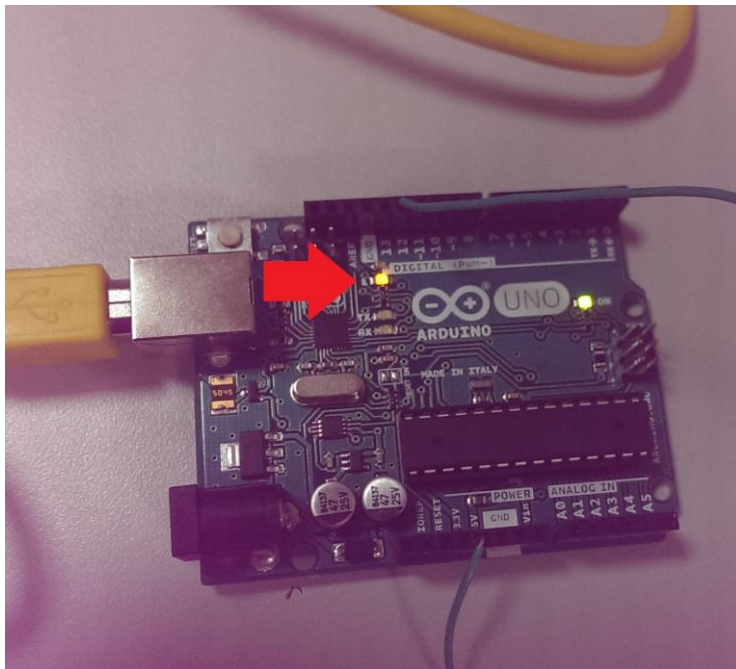
```
LED $
int led = 13;
void setup () {
    pinMode(led, OUTPUT); //LED 13 como salida
    Serial.begin(9600); //Inicializo el puerto serial a 9600 baudios
}

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);
        }
    }
}
```

Se comprueba de que el código es correcto y se sube a la placa. A continuación se utiliza el monitor serie para enviar a la placa una “**H**” si se quiere encender el **LED** y una “**L**” si se quiere apagar.



Un **LED** de la propia placa **Arduino**, que está conectado al **pin 13** era el que se encendía o apagaba cada vez que se le daba la orden.

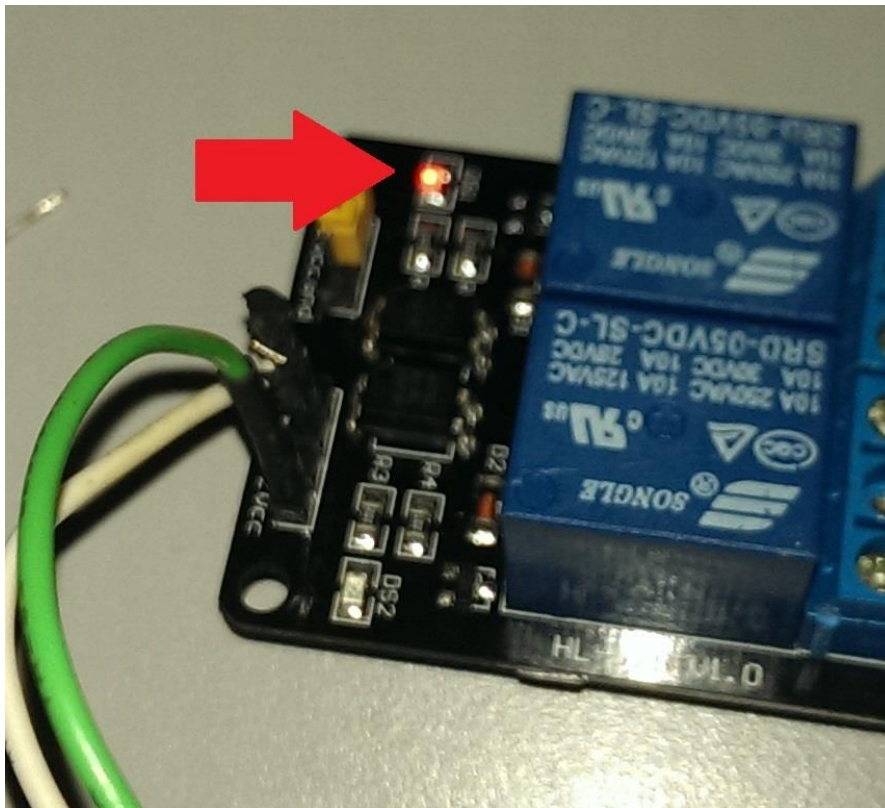


El siguiente punto consiste en utilizar **Python** para mandar comandos a la placa **Arduino**. Para ello, lo primero es instalar la librería de **Python**. Cuando esté instalado, se introduce el siguiente código en un archivo `.py`.

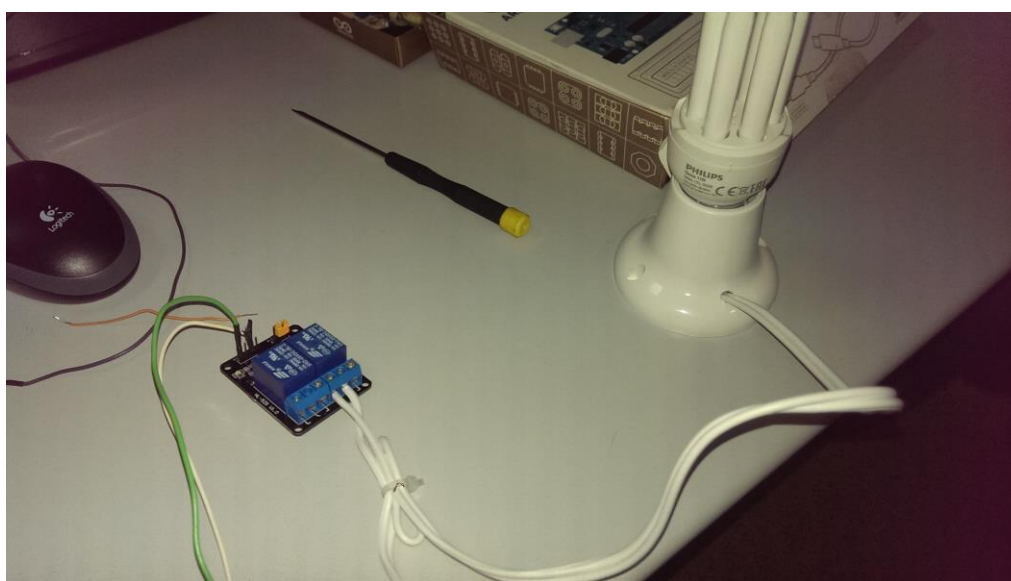
```
led.py x
1 import serial
2
3 arduino = serial.Serial('/dev/ttyACM0', 9600)
4
5 print("Starting!")
6
7 while True:
8     comando = raw_input('Introduce un comando: ') #Input
9     arduino.write(comando) #Mandar un comando hacia Arduino
10    if comando == 'H':
11        print('LED ENCENDIDO')
12    elif comando == 'L':
13        print('LED APAGADO')
14
15    arduino.close() #Finalizamos la comunicacion
```

Utilizando el símbolo del sistema, o terminal, según el **SO**, ejecutamos el código y desde ahí podremos enviar “**H**” para encender el **LED** o “**L**” para apagarlo.

La siguiente modificación fue **utilizar un relé** y comprobar que funcionaba exactamente igual. Utilizando la consola y ejecutando el programa en **Python** se comprueba con un **LED** que tiene incorporado el relé.



La siguiente modificación era conectando una bombilla al relé. Volviendo a utilizar la consola y el programa en **Python**, se comprueba que funcionaba correctamente.



Sesión 2 de laboratorio

Encendido y apagado de un LED mediante un pulsador

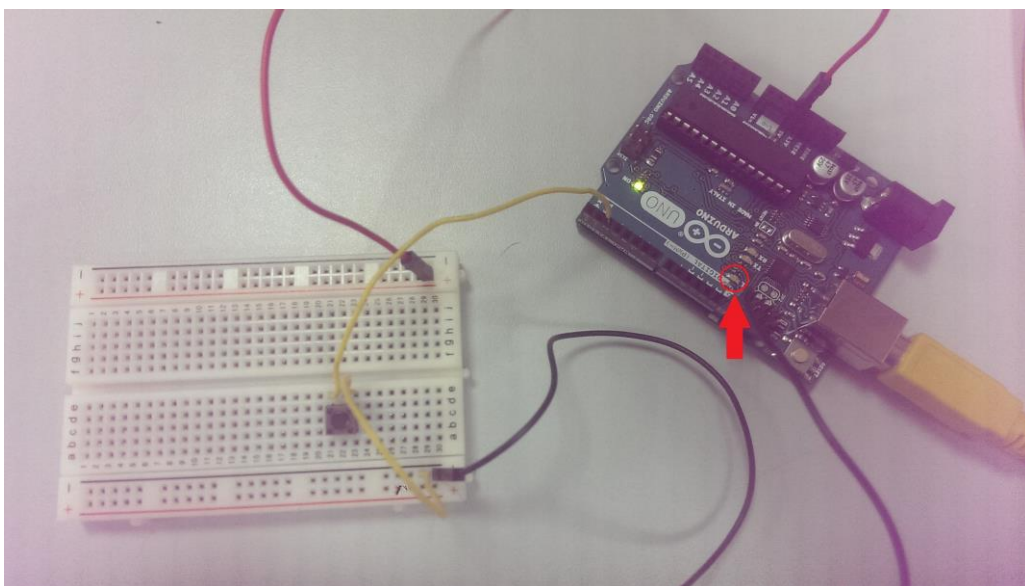
En esta sesión de laboratorio, utilizamos un botón, que conectamos a la **protoboard**, y el **LED** de la placa **Arduino** que está conectado al **pin 13**.

El comportamiento que se quiere conseguir es que cuando se pulse el botón se encienda el **LED**, y cuando se vuelva a pulsar se apague.

En el código que se nos proporcionaba, no tenía este comportamiento, pero bastó con cambiar un parámetro de la función **attachInterrupt()**. Había que cambiar **CHANGE** por **FALLING**. Además, utilizamos la configuración **PULL-UP**.

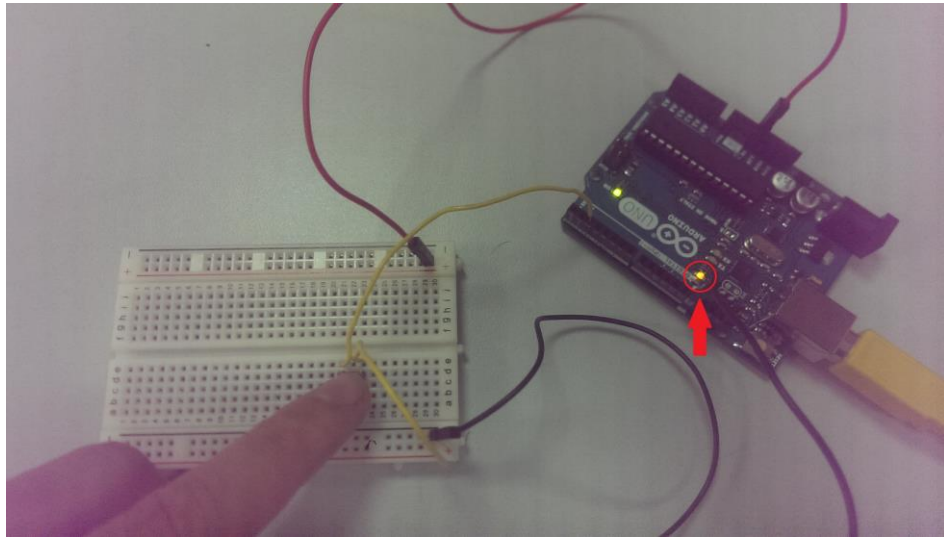
```
LED §  
  
const byte ledPin = 13;  
const byte interruptPin = 2;  
volatile byte state = LOW;  
  
void setup() {  
  pinMode(ledPin, OUTPUT);  
  pinMode(interruptPin, INPUT_PULLUP);  
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, FALLING);  
}  
  
void loop() {  
  digitalWrite(ledPin, state);  
}  
  
void blink() {  
  state = !state;  
}
```

El circuito resultante es el siguiente:



Se puede ver el botón en la **protoboard** y la flecha apuntando al **LED** de la placa que vamos a utilizar.

Cuando se pulsa el botón, el **LED** se enciende:



Y si se pulsa de nuevo, se apaga.

Control de la posición de un servomotor con un potenciómetro

En este punto, se utilizará en la placa **Arduino** un pin que recibirá la señal analógica que proviene del potenciómetro. El código **Arduino** correspondiente al circuito que se va a construir es el siguiente:

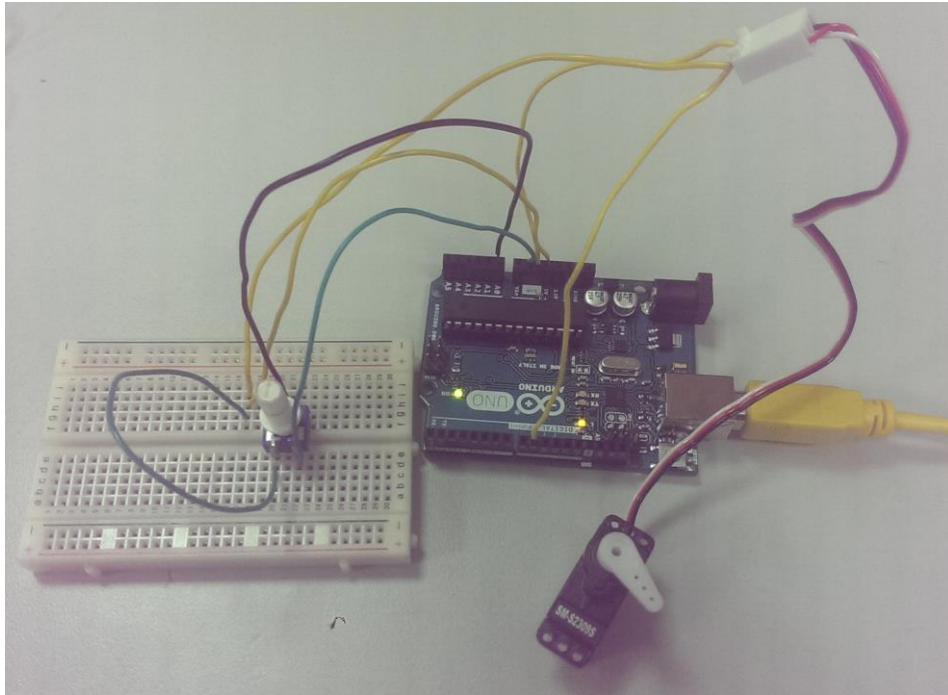
```
#include <Servo.h>
Servo myservo; // Crea un objeto servo

int potpin = 0; // Pin analógico que irá conectado al potenciómetro
int val; // Variable que leerá el valor del pin analógico

void setup() {
  myservo.attach(9); // attaches the servo on pin 9 to the servo object
}

void loop() {
  val = analogRead(potpin); // Lee el valor del potenciómetro (Valor entre 0 y 1023)
  val = map(val, 0, 1023, 0, 180); // Es una escala que se utiliza para que funcione en el servo (Valor entre 0 y 180)
  myservo.write(val); // Modifica la posición del servo según el valor que se ha proporcionado
  delay(15);
}
```


Y el circuito construido en la placa y la **protoboard** es el siguiente.



Cuando girábamos al máximo el potenciómetro, el servo llegaba a su tope, y comenzaba a vibrar. Esto se solucionaría modificando el código y señalando que el servo no llegará al tope, es decir, si el tope es 180° , que solo llegará al 179° , o hasta donde la vibración dejará de existir.

Modificación propuesta

En el siguiente punto, se pedía una modificación del código y circuito anterior, para hacer girar el servomotor desde el ordenador, utilizando un código en **Python**.

Primero se modificó el código **Arduino**, eliminando la parte del potenciómetro, para que no hubiera conflicto con el nuevo circuito.

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

void setup() {
  myservo.attach(9);
  Serial.begin(9600);
}

void loop() {

  if(Serial.available()){

    int angulo = Serial.parseInt();
    myservo.write(angulo);
  }
}
```

Después se creó el código **Python** que se iba a ejecutar:

```
import serial

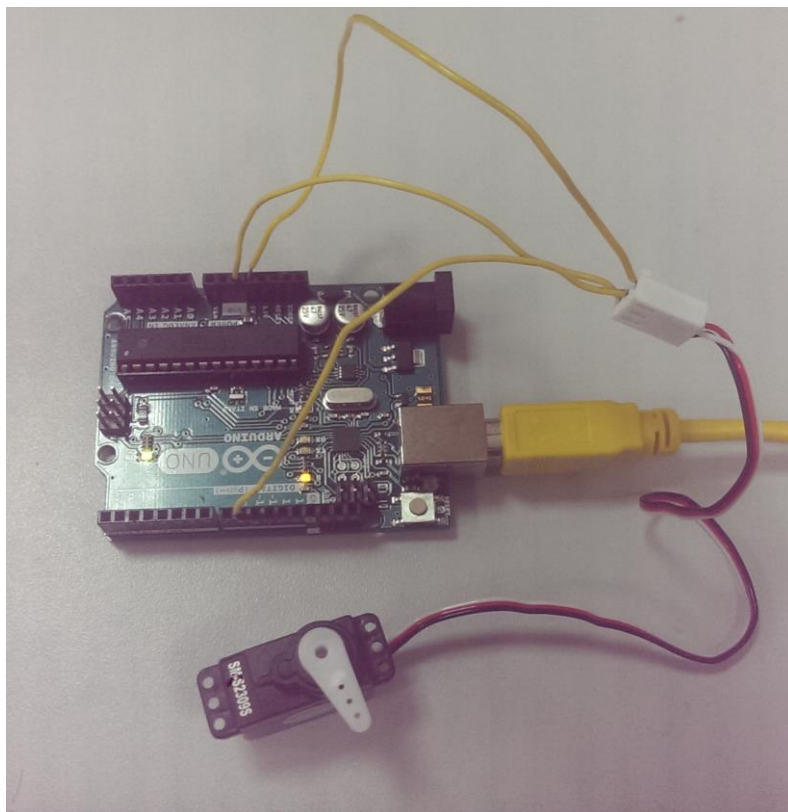
arduino = serial.Serial('/dev/ttyACM1', 9600)

print("Starting!")

while True:
    comando = raw_input('Introduce un comando: ') #Input
    arduino.write(comando) #Mandar un comando hacia Arduino

arduino.close() #Finalizamos la comunicacion
```

Y por último se montó el circuito:



Sesión 3 de laboratorio

Control de la velocidad de giro de un motor de corriente continua con un potenciómetro

En este ejemplo de programación sobre **Arduino**, se pretende **controlar la velocidad de giro de un motor de corriente continua**, y además, el sentido en el que gira.

Lo primero de todo es realizar el código en **Arduino**

```
int pin2=9;    //Entrada 2
int pin7=10;   //Entrada 7
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

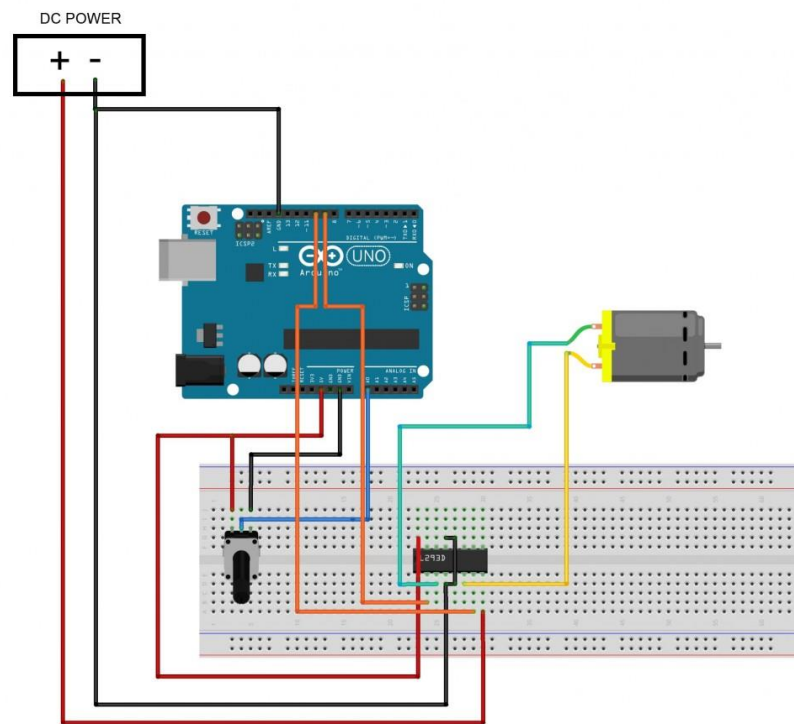
void setup()
{
    //Inicializamos los pins de salida
    pinMode(pin2,OUTPUT);
    pinMode(pin7, OUTPUT);
}

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

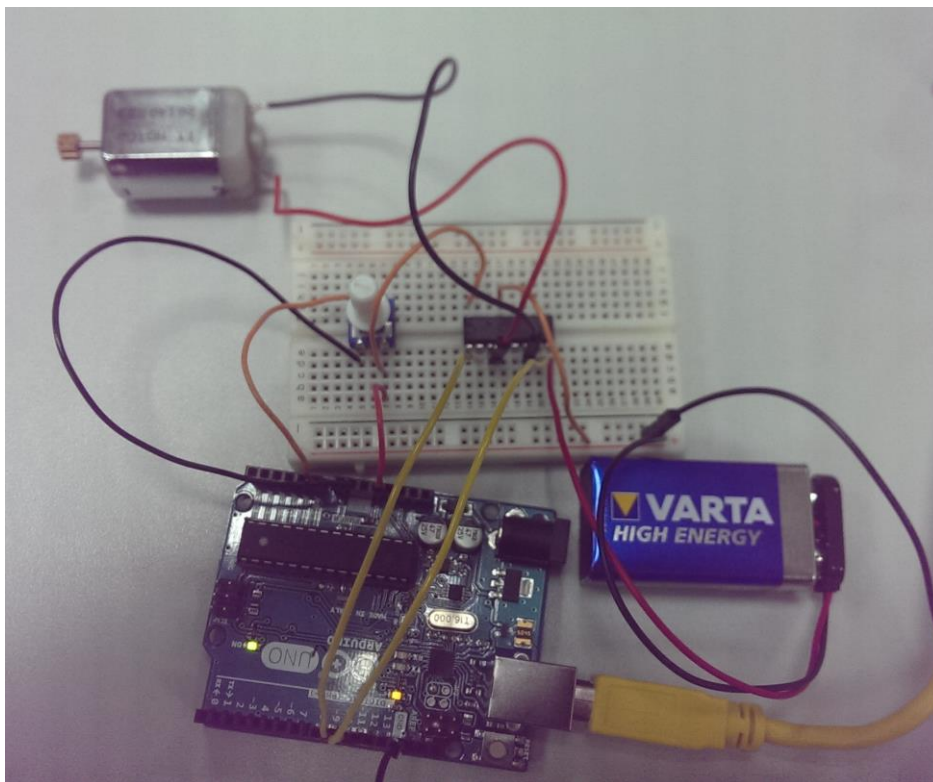
    pwm1 = map(valorpote, 0, 1023, 0, 255);
    pwm2 = map(valorpote, 0, 1023, 255, 0);

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

Una vez realizado el código, lo siguiente es construir el siguiente circuito:



Quedando de la siguiente manera en la **protoboard** y la placa **Arduino**:



Impresión en una pantalla LCD

El siguiente ejemplo consiste en, haciendo uso de una pantalla **LCD**, un potenciómetro, y la placa **Arduino**, mostrar el acrónimo de la asignatura y el nombre del alumno.

Para ello, primero se escribió el código en el entorno de desarrollo Arduino:

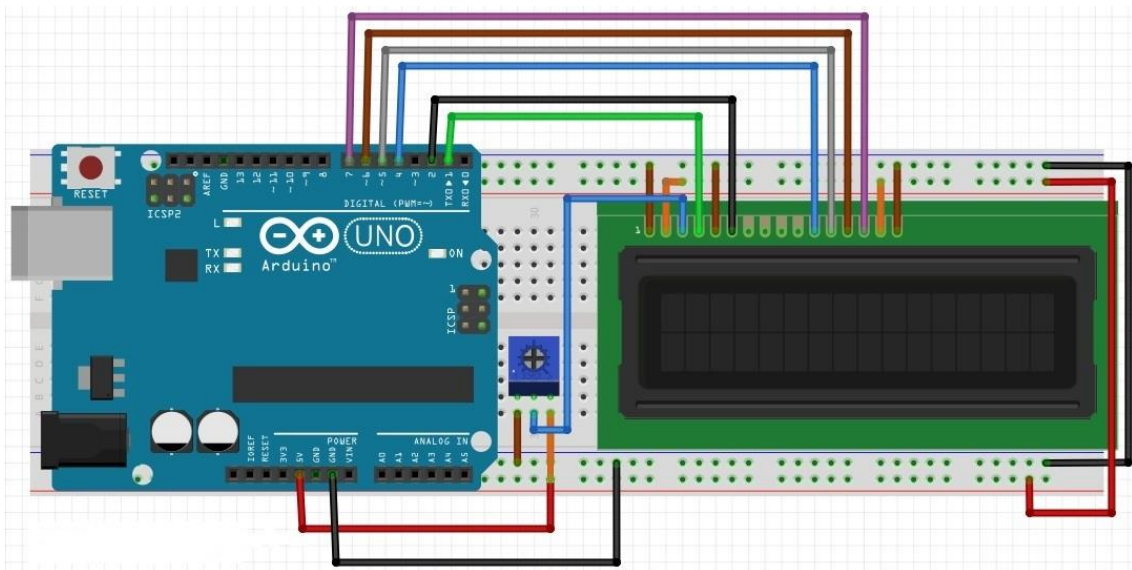
```
#include <LiquidCrystal.h>

LiquidCrystal lcd(1, 2, 4, 5, 6, 7);

void setup() {
  lcd.begin(16,2);
}

void loop() {
  lcd.print("LDH");
  lcd.setCursor(0,2);
  lcd.print("Antonio Manuel");
}
```

Una vez escrito el código, montamos el circuito correspondiente al siguiente esquema:



El resultado es:



Modificación propuesta

Se pide modificar el proyecto de la impresión en pantalla **LCD**, haciendo que imprima información que se le envía desde el ordenador vía serie.

Para ello, hay que hacer las siguientes modificaciones en el código **Arduino**:

```
#include <LiquidCrystal.h>
char c;
String cadena;
LiquidCrystal lcd(1, 2, 4, 5, 6, 7);
void setup() {
  lcd.begin(16,2);
  Serial.begin(9600);
}
void loop() {

  if(Serial.available() > 0){
    delay(100);
    lcd.clear();
    while(Serial.available() > 0){

      c = Serial.read();
      cadena += c;
    }
    lcd.print(cadena);
    c = '\0';
    cadena = "";
  }
}
```

El siguiente punto es crear un código en **Python** para poder enviar la información vía serie. El código es el siguiente:

```
import serial

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

print("Starting!")

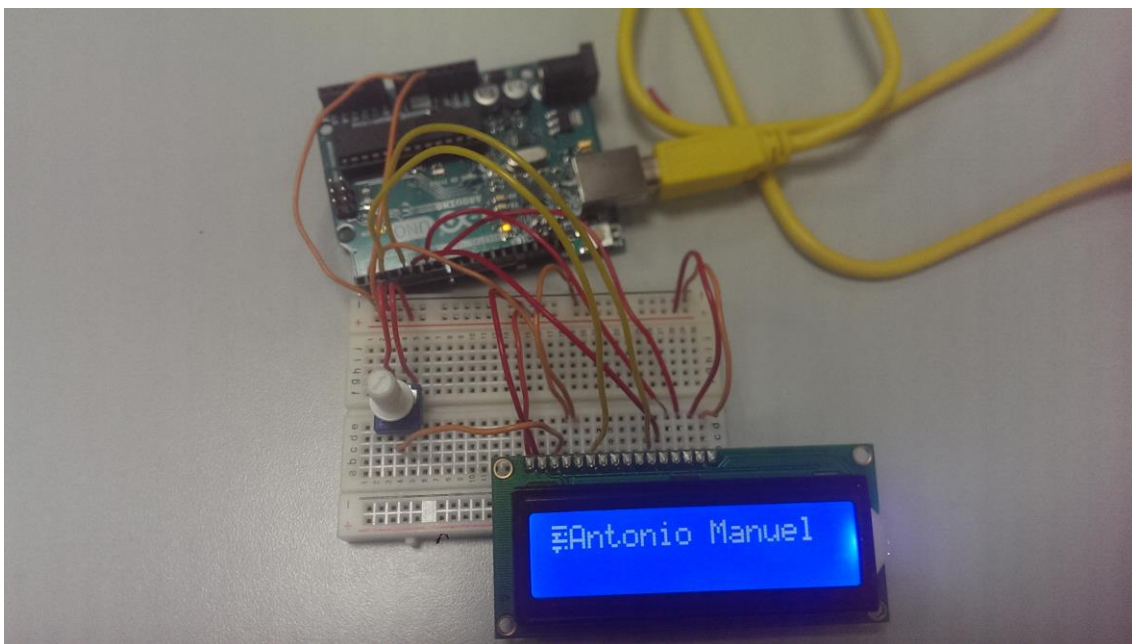
while True:
    comando = raw_input('Introduce un comando: ') #Input
    arduino.write(comando) #Mandar un comando hacia Arduino

arduino.close() #Finalizamos la comunicacion
```

Haciendo uso del terminal, enviamos la información:

```
practicass@mcr-95:~/Escritorio$ python lcdpy.py
Starting!
Introduce un comando: Antonio Manuel
Introduce un comando: █
```

Y el resultado es el siguiente:



Diseñando termómetro con pantalla LCD y sensor de temperatura

En el siguiente diseño, se va a utilizar un sensor de temperatura **tmp36GZ** y se **mostrará en la pantalla del LCD la temperatura que capta el sensor**. Primero, se diseña el código:

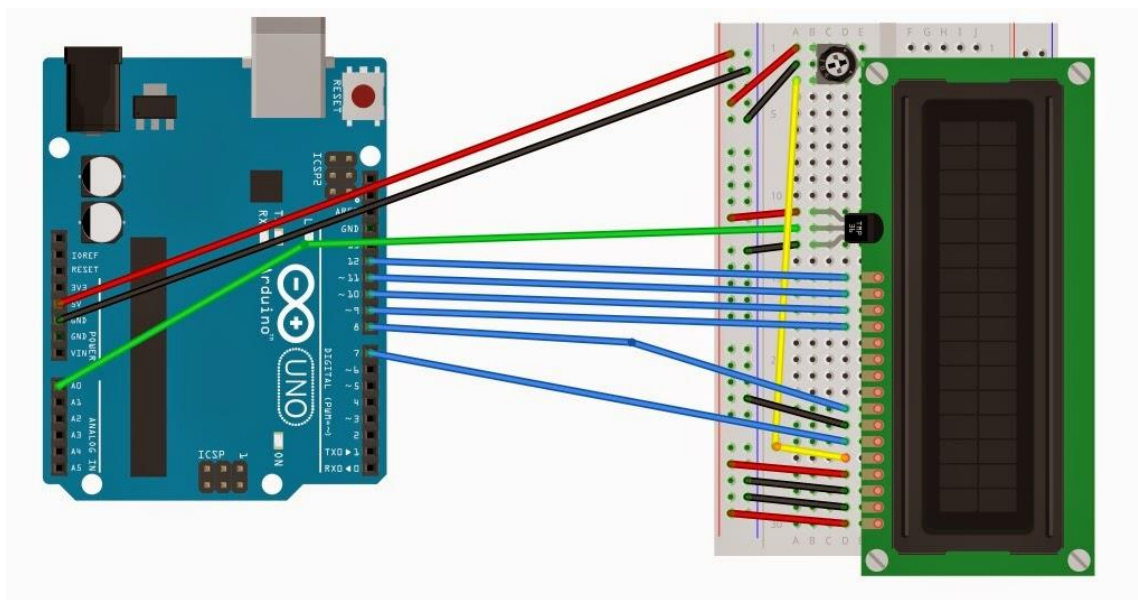
```
#include <LiquidCrystal.h>
LiquidCrystal lcd(7, 8, 9, 10, 11 , 12);
const int sensor = A0;

void setup() {
  Serial.begin(9600);
}

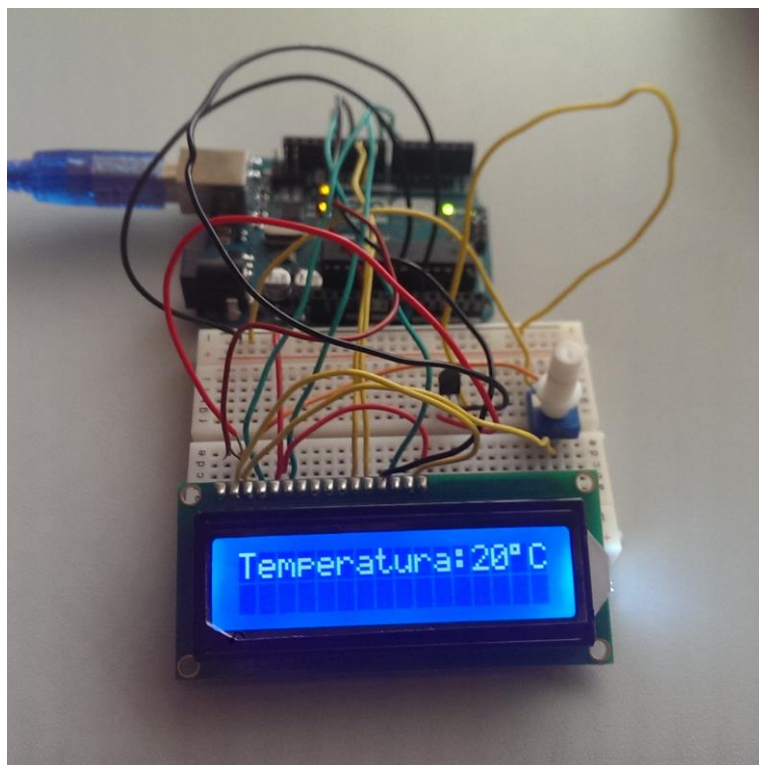
void loop() {
  int sensorVal = analogRead(sensor);
  float voltage = (sensorVal/1024.0) * 5.0;
  Serial.print(" Temp. Grados: ");
  int temperatura = (voltage - .5) * 100;
  Serial.println(temperatura);
  lcd.begin(16, 2);
  lcd.setCursor(0,0);
  lcd.write("Temperatura: ");
  lcd.setCursor(12,0);
  lcd.print(temperatura);
  lcd.setCursor(14,0);
  lcd.write((char)223);
  lcd.setCursor(15,0);
  lcd.write("C");
  delay(2000);
}
```

El sensor de temperatura que se está utilizando, el **tmp36GZ**, emite un voltaje en función de la temperatura, pero **Arduino** no proporciona un voltaje como valor de la entrada analógica. Para ello se necesita un convertidor de analógico a digital. Se recibe un valor entre **0 y 1023**, proporcional al voltaje recibido. Si se quiere operar con los valores del voltaje, se necesita dividir el valor de la lectura de los **1024** valores posible y multiplicar por 5, ya que las entradas analógicas de **Arduino** solo pueden medir valores entre 0 y 5V.

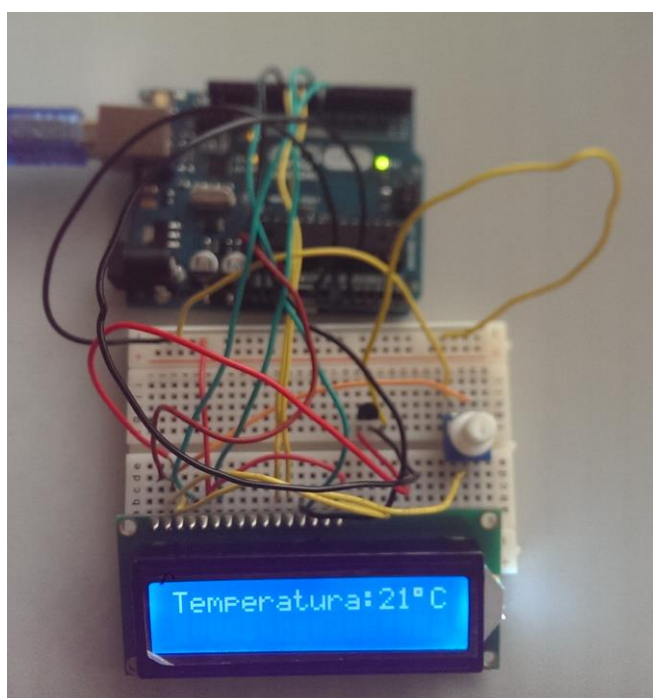
Ahora se procede a construir el circuito según el siguiente esquema:



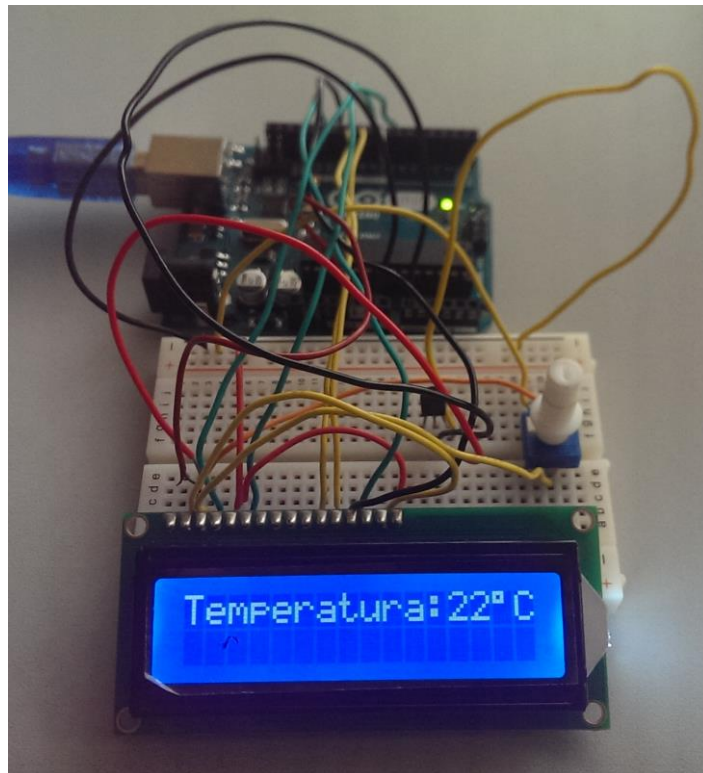
Al final tenemos el resultado:



Los 20° C era la temperatura que había en el momento de la foto. Justo después comencé a tocar el sensor con el dedo para subir la temperatura, obteniendo:



Y después:



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

En este último ejercicio, se pide diseñar un contador que vaya de 0 a 59, que se vaya incrementando 1 cada segundo, y se muestre en un **display 7-segmentos**. Lo primero es diseñar el código:

```
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;

void setup() {
  for (int seg = 0; seg < 8; seg++) {
    pinMode(segPins[seg], OUTPUT);
  }
  pinMode(disp1, OUTPUT);
  pinMode(disp2, OUTPUT);
}

void loop() {
  // Cálculo correcto de los datos dat1, dat0 --> desde 0 0 hasta 5 9
  dat0=cont / 10;
  dat1=cont % 10;
  refresh(dat1,dat0);
  if(cont<59){
    cont=cont+1;
  }else{
    cont=0;
  }
}
```

Esta parte del código se emplea para transformar el dato obtenido del cálculo de los datos **dat1**, **dat0**, en el código de siete segmentos para poder escribirlo en el **display**:

```
// Función write_data(dato): Transforma el dato en su código siete segmentos para escribirlo en el display
void write_data (int arg) {

    switch (arg) {
        case 0:
            write7seg(0x7e);
            break;
        case 1:
            write7seg(0x30);
            break;
        case 2:
            write7seg(0x6d);
            break;
        case 3:
            write7seg(0x79);
            break;
        case 4:
            write7seg(0x33);
            break;
        case 5:
            write7seg(0x5b);
            break;
        case 6:
            write7seg(0x1f);
            break;
        case 7:
            write7seg(0x70);
            break;
        case 8:
            write7seg(0x7f);
            break;
        case 9:
            write7seg(0x73);
            break;
    }
}
```

La función **refresh** es necesaria para evitar el parpadeo que produce el **display**. Es necesario darle una frecuencia de parpadeo adecuada para que el ojo humano vea los números que se muestran en el **display** sin problemas.

La segunda función, **write7seg** escribe el valor que se quiere mostrar en el **display**.

```
// Función refresh: Duración total de ejecución de refresh: tiempototal.
//Se van intercambiando los displays (displ con dat0 y disp2 con dat1) a
void refresh( int dat1, int data0) { //una frecuencia que evite el parpadeo (j--> numero de veces que se activan ambos displays)
    int tiempo_refresco = tiempototal/(2*j);
    // Bucle de activación de los displays. El bucle se ejecuta j veces.
    for(int i=0;i<=j;i++){ //Para escribir en los displays se llama a la función write_data (dato)
        digitalWrite(displ,0);
        digitalWrite(displ2,1);
        write_data(dat1);
        delay(tiempo_refresco);
        digitalWrite(displ,1);
        digitalWrite(displ2,0);
        write_data(data0);
        delay(tiempo_refresco);
    }
}
// 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;
    }
}
```

En la siguiente línea de código, el número 2 sirve para que el contador vaya incrementando a la velocidad normal de los segundos. Si cambiáramos el 2 por un 6, el contador iría mucho más rápido.

La variable *j*, sirve para indicar la frecuencia de refresco del **display**. En el ejemplo, tenemos que *j* = 40. Con este valor, el ojo humano apenas nota el parpadeo del **display**, sin embargo, la cámara de un teléfono móvil si lo captaría. Para que la cámara no lo capte es suficiente con cambiar el valor de *j* = 80.

```
int tiempo_refresco = tiempototal/(2*j);
```

Al final tenemos como resultado:

