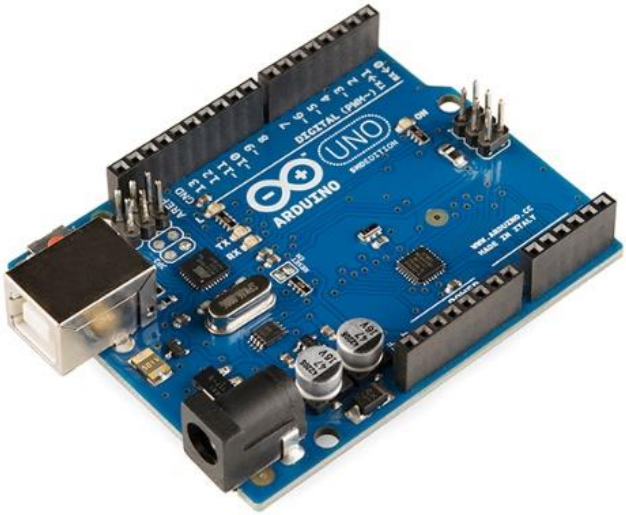
Sensores y actuadores



httphttps://commons.wikimedia.org/wiki/File:Arduino_Uno_-_R3.jpg

Práctica 2 – Teoría (v1.7.3 septiembre 2022)

Software para robots

Cristian González García

gonzalezcristian@uniovi.es

Basado en el material original de Jordán Pascual Espada

Índice

1. Introducción

El objetivo principal de esta práctica es utilizar las entradas y salidas de Arduino para controlar otro tipo de elementos más complejos (sensores, actuadores y elementos de interfaz de usuario).

Los **sensores** nos permiten obtener información del entorno. Para utilizar un sensor desde Arduino se suele utilizar al menos una entrada digital o analógica, dependiendo del tipo de sensor. Algunos sensores pueden requerir el uso de varias salidas, o de varias entradas, para enviar parámetros que configuren su funcionamiento o sean parte de él. Un ejemplo es el de ultrasonidos, que necesita primero enviar una onda, que será la que utilice para detectar la distancia.

La mayor parte de **actuadores** se gestionan mediante el uso de salidas. Los actuadores pueden ser: pantallas, motores, LEDs, etc. Estos pueden utilizar más de una salida, analógicos o digitales, dependiendo de la complejidad del actuador y el modelo.

Todos los ejercicios se pueden realizar utilizando el *Shield* de sensores o sin él, utilizando los pines equivalentes del Arduino. El *Shield* lo que proporciona es una pequeña «facilidad» en la conexión de sensores y actuadores y proporciona más pines GND y V.

2. Sensores

Sensor de luz

Sensor de luz: este tipo de sensores deben conectarse a un **pin analógico**, pues nos devolverá más de 2 valores.

Retorna un valor que suele oscilar entre 0 – 620: a mayor valor retornado mayor cantidad de luz está detectando.

En este modelo:

- el **pin negro va a GND**, pues es el negativo.
- el **pin central y rojo a V**, a la potencia.
- el **pin restante y de otro color** (azul o blanco, depende del modelo) **a un pin analógico de lectura**. Es decir, a un pin analógico entre A0 y A5 en el Arduino UNO**.**

Sensor PIR

Sensor de movimiento pasivo, PIR, o pasivo infrarrojo: permite detectar movimiento cercano ya que detecta la diferencia de calor emitido por las fuentes de energía, respecto al espacio de alrededor. Por ejemplo, detecta el calor emitido por el cuerpo humano o el de los animales. Es pasivo debido a que recibe radiaciones y no las emite. El sensor está recubierto por una lente de Fresnel que le ayuda a incrementar el área de funcionamiento.

Cuando se instalan, necesitan «aclimatarse» a las radiaciones del ambiente, pues reciben radiaciones de todo su alrededor. Una vez estén «acostumbrados» son capaces de detectar las variaciones en ese ambiente, como cuando una persona entra en la habitación, ya que al entrar se modifica la radiación infrarroja del ambiente.

Cuando detecta movimiento mantiene una salida de 5V, mientras que cuando no hay movimiento mantiene una salida de 0V (algunas marcas funcionan de la forma contraria). Para detectar movimiento, el sensor utiliza un detector de niveles de radiación infrarroja debido a que la mayor parte de cosas que se mueven generan un pequeño nivel de radiación infrarroja (calor).

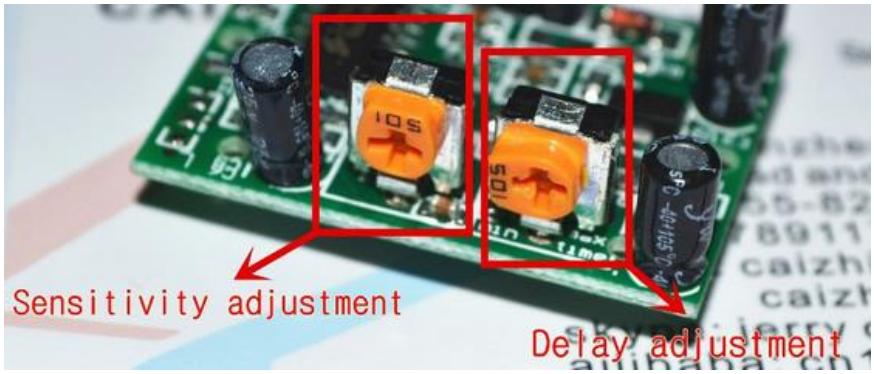
Dependiendo del tipo de sensor, este puede tener diferente distancia de detección y diferente ángulo de detección. Unos valores típicos para un sensor de este tamaño son 7 metros y 110°.

En este modelo:

- el **pin GND** es el negativo.
- el **pin VCC**, a la potencia.

- el **pin OUT va a un pin digital de lectura**. Es decir, a un pin digital entre 2 y 12 en el Arduino UNO**.**

Algunos sensores como este incluyen cierta capacidad de calibración. En este caso, tenemos dos tornillos en la parte inferior que nos permiten ajustar el grado de sensibilidad del sensor y el retardo entre detecciones.



Notas importantes:

Los reflejos o pequeños movimientos de la mesa pueden hacer que detecte movimiento. Recordad, que el sensor tiene que «aclimatarse» al ambiente.

Puede ser que la lente Fresnel se caiga del sensor, pues el acoplamiento entre ambos es bastante bajo. Simplemente hay que volver a colocarla haciendo que encaje y se asiente correctamente sobre la palca del sensor.

Sensor de vibración

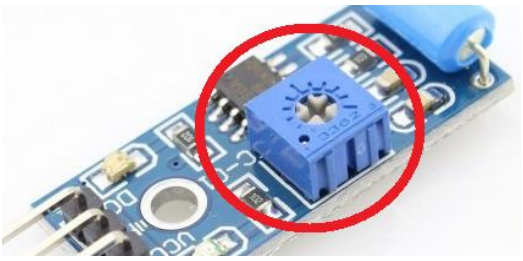
Sensor de vibración SW-400: permite detectar vibraciones por encima de un umbral. El umbral se puede configurar de forma física ajustando el tornillo.

Se suelen utilizar en aplicaciones industriales para medir la fiabilidad de motores, engranajes, bombas, máquinas, etc. De esta manera, se pueda tener información sobre un mal comportamiento para repararlo cuanto antes.

Cuando no detecta vibración retorna una salida de 0V (LOW) y cuando la detecta retorna 5V (HIGH).

La patilla **D0** se conecta a un pin digital, **VCC** a un voltaje, y **GND** que es la patilla central a la toma tierra/negativo**.**

Ajuste: si giramos el tornillo hasta su límite en sentido antihorario, lo colocamos en su mayor nivel de sensibilidad y podrá detectar vibraciones muy leves. Cuanto más giremos el tornillo de forma horaria más reduciremos la sensibilidad y más fuertes deberán ser las vibraciones para ser detectadas.



Sensor de proximidad por infrarrojos

Sensor de proximidad por infrarrojos: permite detectar elementos colocados delante del sensor. La distancia de detección se puede configurar de forma física ajustando el tornillo.

Está compuesto por un emisor de infrarrojos que emite pulsos de forma continua y por un detector de infrarrojos que detecta cuando la luz infrarroja rebota contra un objeto, captando ese rebote. De esta forma sabemos que hay algo delante del sensor. En función del ajuste, detectará que hay objetos delante de él a mayor o menor distancia.

Cuando no detecta ningún elemento retorna una salida de 5V (HIGH), y cuando detecta un elemento retorna 0V (LOW). Este es el **funcionamiento opuesto de los sensores anteriores**, cuidado.

El LED que está cerca de la salida **OUT** se enciende en rojo cuando detecta algo.

La patilla **OUT** se conecta a un pin digital, la **VCC** a un voltaje y la patilla **GND** a una tierra (G) del Arduino.

Ajuste: si giramos el tornillo hasta su límite en sentido antihorario, lo colocamos en su menor nivel de detección y no detectará absolutamente nada. Fijaros que, en este caso, es totalmente al contrario que el sensor de vibración. Esto depende del fabricante del sensor.

Cuanto más giremos el tornillo de forma horaria, más aumentaremos el nivel de detección en centímetros. Primero, detectará elementos que estén únicamente a 0-2 cm. Según vayamos girando el tornillo podrá llegar a detectar a 0-8 cm, aproximadamente.

Sensor de ultrasonidos HC-SR04

Sensor de ultrasonidos SR04: detecta la distancia a la que se encuentra un objeto por medio del uso de ultrasonidos.

Este sensor emite y recibe el rebote de un ultrasonido. Luego, si calculamos el tiempo transcurrido entre estas dos acciones es posible saber si la onda colisionó con algún objeto y su distancia.

Dependiendo del fabricante y el tipo del sensor, estos pueden tener más o menos precisión y alcance: esta versión **permite medir distancias de 2cm y 4m con una precisión aproximada de 3mm**, teóricamente, y **dependiente del modelo**. No obstante, en la práctica, cuanto más cerca de los extremos se quiera medir, más problemas dará, aunque esto varía según el sensor.

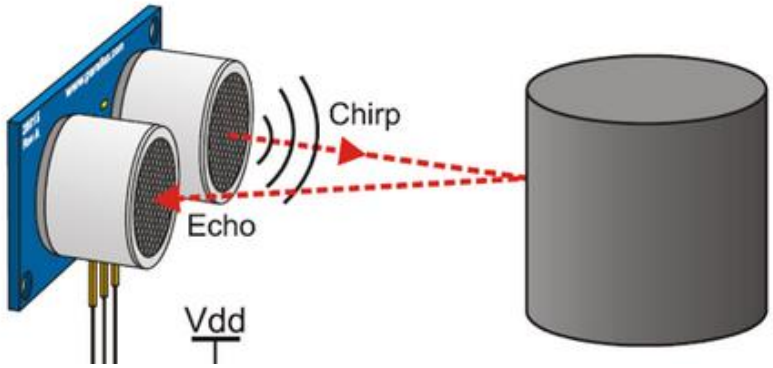
El sensor tiene cuatro pines:

- **VCC** va a la potencia.
- **GND** va al negativo.
- **Trig** se conecta a un ping digital y se encargará de enviar el pulso de ultrasonidos.
- **Echo** se conecta a otro pin digital y se encargará de recibir el eco de dicho pulso.

Entre pulsos hay que esperar al menos 10-20 microsegundos para evitar problemas. Además, **en el primer disparo hay que ponerlo a LOW y esperar al menos 5 microsegundos para asegurar realizar un «disparo limpio»**.

Cálculo de la distancia

1. Para saber a qué distancia está un objeto, deberemos calcularla. Para ello, sabemos que un sonido se mueve a 0,034 cm/microsegundo:
    a. Velocidad del sonido: 343,2 m/s -> 343 m/s * 100cm/m * 1s/1000000 ms = 0,03432 cm/ms
2. Hay que tener en cuenta, que la onda recorre el doble de distancia de a la que realmente está el objeto, pues la onda la enviamos, choca con el objeto, y vuelve. Así pues, hay que dividir entre 2 el cálculo:
    a. 0,03432 / 2 = 0,01716
3. Problemas:
    a. Podría pasar que se false la lectura por los rebotes que pueda dar el pulso contra varios objetos o contra el propio objeto si no impacta de forma directa o tiene ángulos.
    b. Si enviamos **dos pulsos muy seguidos** que recibamos el primero después de enviar el segundo, lo que dará una lectura errónea. Para esto, hay que tener en cuenta que el sonido se desplaza a 0,03432 cm/ms (**1 segundo son 34,32 cm**), así que a distancias más largas necesitará mayor tiempo de espera.



Notas importantes:

En ocasiones el sensor de sonidos deja de emitir mediciones o solo emite mediciones erróneas. En ese caso, hay que desconectar el USB y volver a conectarlo.

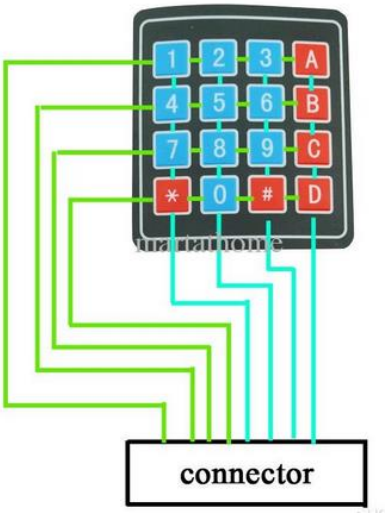
Algunas veces, algunos sensores no bajan de 5 cm de distancia mínima, o bien, devuelven siempre 0 en la primera petición. Esto puede ser problemas del propio sensor. Una posible solución es desconectar el USB y volver a conectarlo. Si sigue dando ese problema hay que hacer un «apaño» por software.

Hay q tener cuidado con los valores del tiempo de la función millis(), pues pueden llegar a ser muy grandes. En este caso, si se ponen como enteros, podría haber un *overflow* y pasar a ser negativos o muy bajos, lo que puede ocasionar problemas. Utilizad entonces el tipo de variable *float* o *double*.

Algunos modelos de sensor de ultrasonidos introducen demasiado «ruido». En estos casos, puede ser una buena estrategia realizar un conjunto de mediciones muy seguidas y tomar como válida la mediana (http://playground.arduino.cc/Main/Average).

Hay que tener en cuenta que, dependiendo de la orientación, la onda puede rebotar sin retornar al lector de ultrasonidos. En este caso, el sensor pensará que no hay nada delante. Es justo por eso por lo que los sensores de ultrasonidos suelen colocarse en servomotores, sobre todo en los robots móviles, y así poder lanzar la onda en varias direcciones y tener mayores posibilidades de éxito.

Teclado



Teclado matricial: los teclados matriciales permiten capturar pulsaciones en una matriz de botones utilizando el indicador de la fila y la columna pulsada.

Para detectar la pulsación se conectan cuatro pines como entradas a las columnas y cuatro pines como salidas a las filas, o viceversa.

El modo de conexión más común es utilizar 8 pines digitales para conectarlo, pero esto, en un Arduino UNO nos dejaría solo con 3 pines digitales usables libres para el resto de los elementos que quisiéramos utilizar. Por eso, para evitar este problema, podemos optar por conectar cuatro cables, por ejemplo, los primeros de la izquierda (verdes) a los pines digitales (2 - 5) y los otros cuatro cables (azules) a los pines analógicos (A0 - A3). Si los conectáis del revés, cambiara la asignación de la matriz de símbolos y sería otra diferente.

3. Actuadores

Pantalla de 7 segmentos y 4 dígitos

Pantalla de 7 segmentos y 4 dígitos: permite representar valores de cuatro dígitos.

Este componente incluye un chip que permite controlar la pantalla haciendo uso únicamente de una entrada y una salida digital.

Este modelo también permite regular el brillo de la luz emitida.

Las conexiones necesarias son:

- **CLK (Pin Digital PWM).**
- **DIO (Data IO, Pin Digital).**
- **VCC (5V).**
- **GND (GND).**

Librería:

- TM1637
- https://github.com/avishorp/TM1637

Servomotor 180º

Servomotor 180°: es un pequeño motor de corriente continua que incluye en su interior una especie de potenciómetro que le permite saber su posición exacta y controlarla.

Gracias a esta particularidad, podemos hacer que el motor se mueva a un ángulo exacto. Existen servomotores de diferentes tipos: unos permiten rotar 360° (rotación continua) y otros 180° o menos, como pueden los de 60°, por ejemplo.

A parte del ángulo de rotación soportado, los servomotores tienen otras características interesantes como el peso que pueden mover o la velocidad a la que se mueven.

El servomotor tiene tres pines:

- **Rojo** que se conecta a **5V** (aunque podría ser más).
- **Negro** o **marrón** que se conecta a **GND**.
- **Blanco** o **naranja** que se conecta a un **pin digital PWM.**

Más información:

- https://www.arduino.cc/en/Reference/Servo
- https://github.com/arduino-libraries/Servo/blob/master/src/Servo.h
-

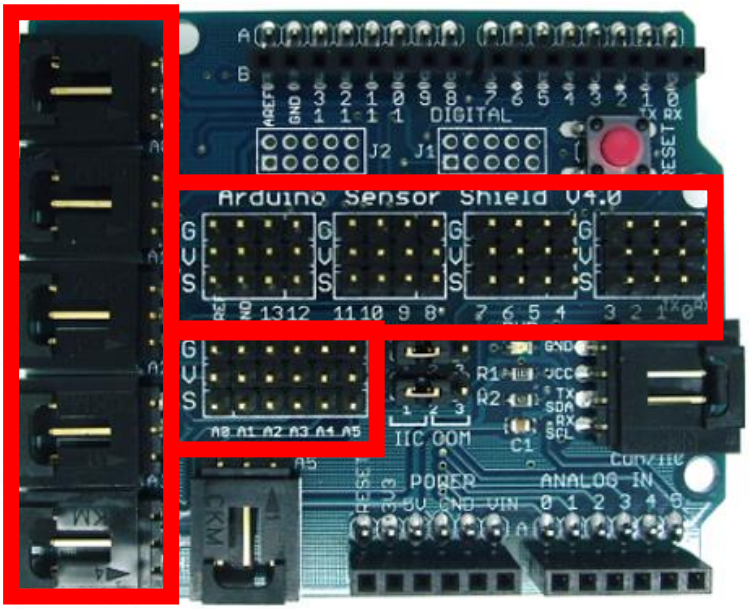
4. Arduino

Shield

Este es el **Arduino Sensor Shield v4.0**. Este *shield* nos permite extender la funcionalidad del Arduino y que nos sea más fácil conectar sensores, sobre todo si estos son los que vienen con un cable entrecruzado inseparable de 3 pines.

Para acoplarlo a la placa Arduino, los pines machos del *shield* deben coincidir perfectamente con los mismos pines hembra del Arduino. Es decir, el pin A0 del *shield* debe enchufarse en el pin A0 del Arduino, y así sucesivamente.

La característica más destacada de este *shield* es la existencia de un pin de tierra (G) y uno positivo (V) por cada pin digital (S superior) y analógico (S inferior). Además del tipo de entrada del lateral izquierdo para los pines analógicos, siendo esta diferente a las del Arduino. El resto del funcionamiento es exactamente igual que en el Arduino.



Toma de tiempos

Como en otros lenguajes de programación, Arduino permite tomar tiempos, Para ello, hay que utilizar la función millis();

Cuando se almacena el valor de esta función en una variable hay que tener en cuenta el tipo de variable, pues es posible que sea muy grande y provoque un *overflow*. Esto sucede si se utilizan variables de tipo `int`, haciendo que se desborde y el número pase a ser negativo y pase de un valor alto a otro bajo rápidamente, ocasionando un mal funcionamiento del programa. Se recomienda utilizar el tipo `long` o `double`.

Incluso, podemos utilizar la versión sin signo de algunos tipos de datos, que se declara escribiendo la palabra reservada `unsigned` delante del tipo de datos. Tenemos dos: `unsigned int` y `unsigned long`. Esto permite usar la capacidad de ese tipo de dato, pero solo para números positivos, doblando así la capacidad de números positivos del tipo de dato.

Además, si se deja funcionando de seguido sobre aproximadamente 50 días, la función hará *overflow* y volverá a cero.

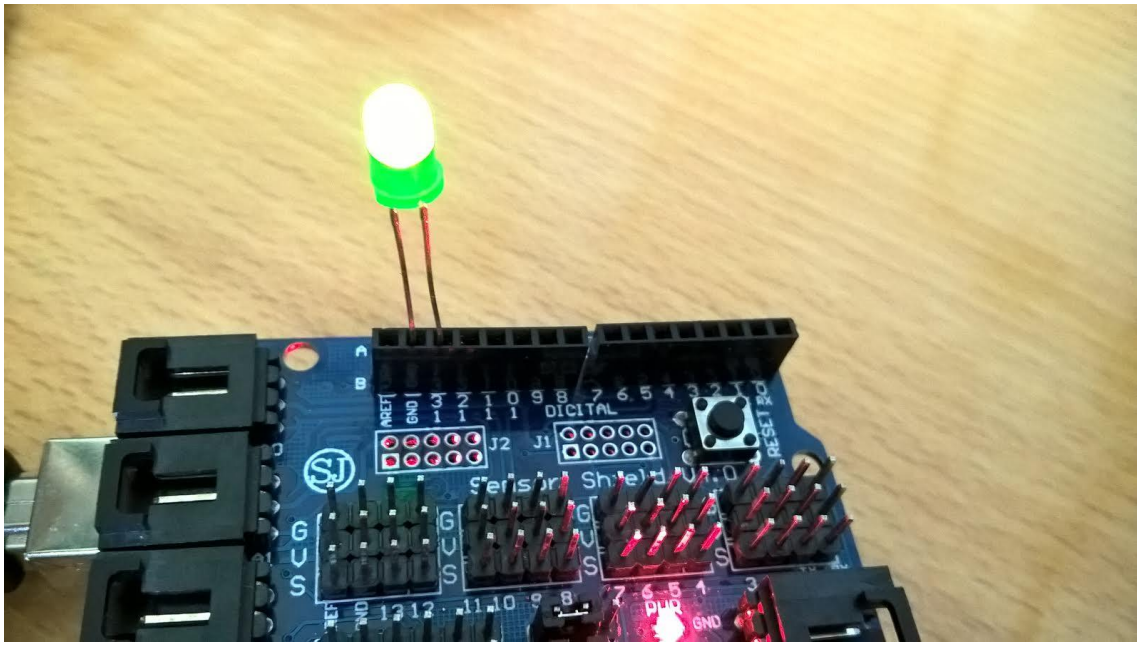
Más información en: https://www.arduino.cc/reference/en/language/functions/time/millis/

Ejemplo de su uso:

```
int led13 = 13;
double inicioCuentaTiempo;

void setup() {
  Serial.begin(9600); // Iniciar el Serial
  pinMode(led13, OUTPUT);

  // Instante de tiempo que empieza a contar!
  inicioCuentaTiempo = millis();
}

void loop() {
  if (millis() - inicioCuentaTiempo  >= 5000){
      // Tiempo actual - inicio Cuenta tiempo >= 5000
      // Han pasado 5 segundos, enciendo el led
      digitalWrite(led13, HIGH);
  }
}
```

Pin 13 del Arudino

El pin 13 del Arudino tiene una resistencia de 220Ω integrada, así como un pequeño LED cerca de este pin. También, dispone de un pin GND justo a su izquierda, por lo que podemos conectar un led de forma rápida para probar si el Arduino está encendido y funcionando, o si el *Shield* que hemos puesto está bien conectado, o probar si un LED funciona o no. Recordad, la patilla larga del LED es el positivo, así que irá al pin 13.



Importar librería

Hay dos formas de importar una librería en Arduino.

Gestor de librerías

La primera de todas es ver si existe en el repositorio de Arduino. Para ello, en el IDE, deberemos ir a Herramientas -> Administrar bibliotecas… (Ilustración 1).

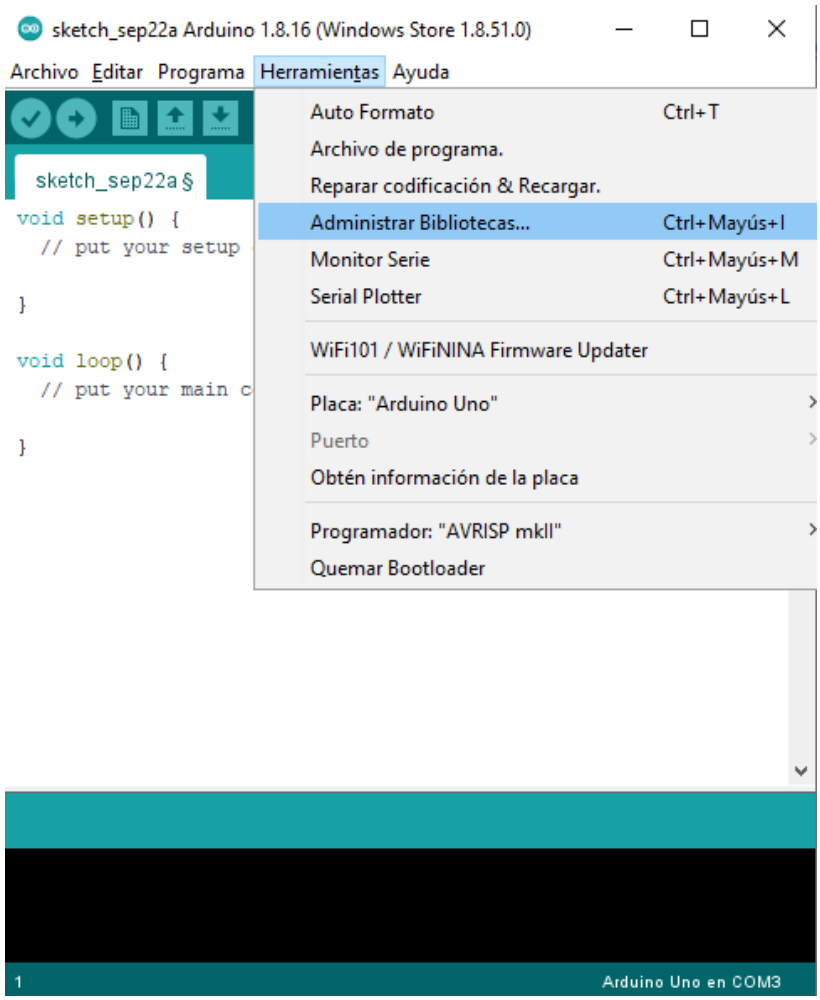


Ilustración 1 Administración de bibliotecas en el IDE de Arduino

Una vez en el gestor (Ilustración 2), tendremos acceso a todas y podremos realizar búsquedas por:

- Tipo: actualizable, instalado, Arduino, compañero, recomendado, contribución, retirado.
- Tema: comunicación, procesando datos, datos almacenados, control de dispositivos, pantalla, otro, sensores, señal entrada/salida, temporizador, sin categoría.
- Palabra: búsqueda a partir de la palabra introducida.

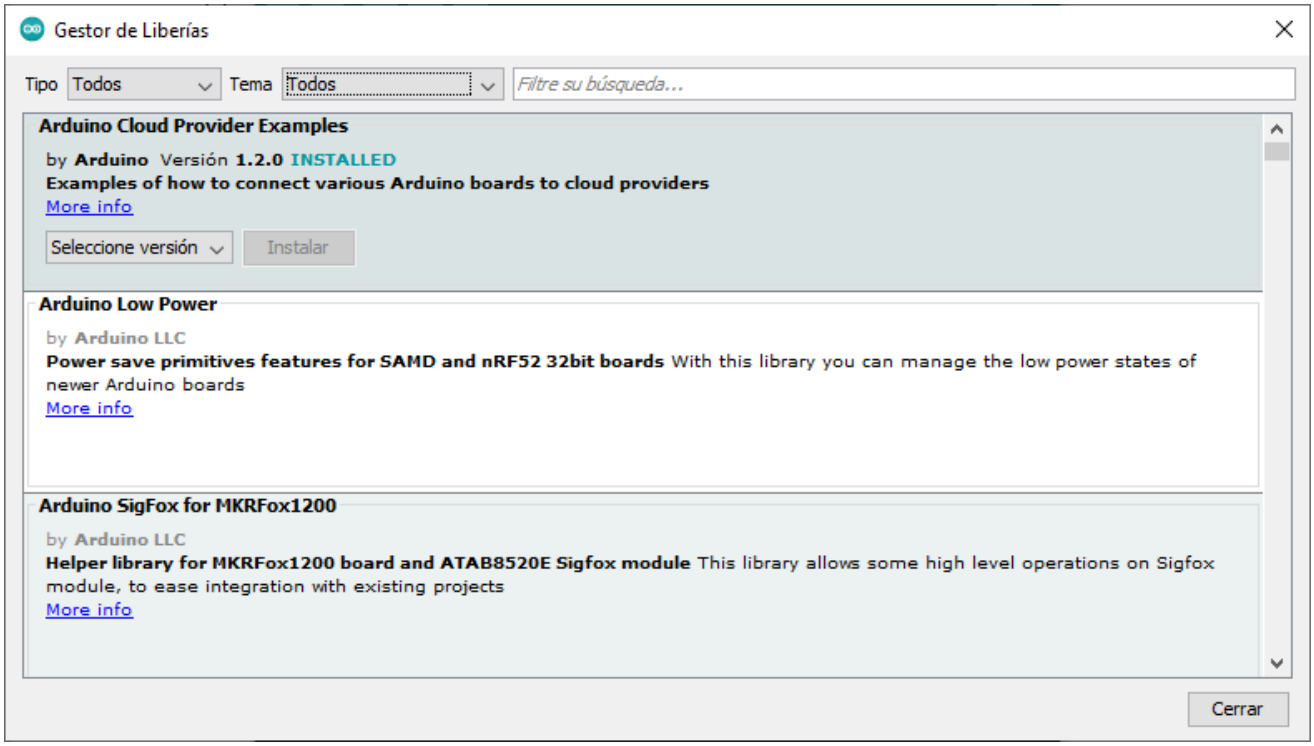


Ilustración 2 Gestor de bibliotecas

Además, permite instalar una versión en concreto si se requiere, la última, actualizar las que ya se tienen instaladas, etc.

Manualmente

En el caso de que la librería no exista en el gestor, podemos añadir de forma manual. Por ejemplo, si quisiéramos añadir la librería para manejar el teclado (KeyPad), tendremos que descargarla de: https://cdn.shopify.com/s/files/1/0557/2945/files/KeyPad_bb5610fd-0c4c-4b0e-8c5e-1e5b3f83f47f.zip?4187195515480435262

Una vez descargada, vamos a **Programa -> Include Library -> Add. Zip Library** (Ilustración 3).

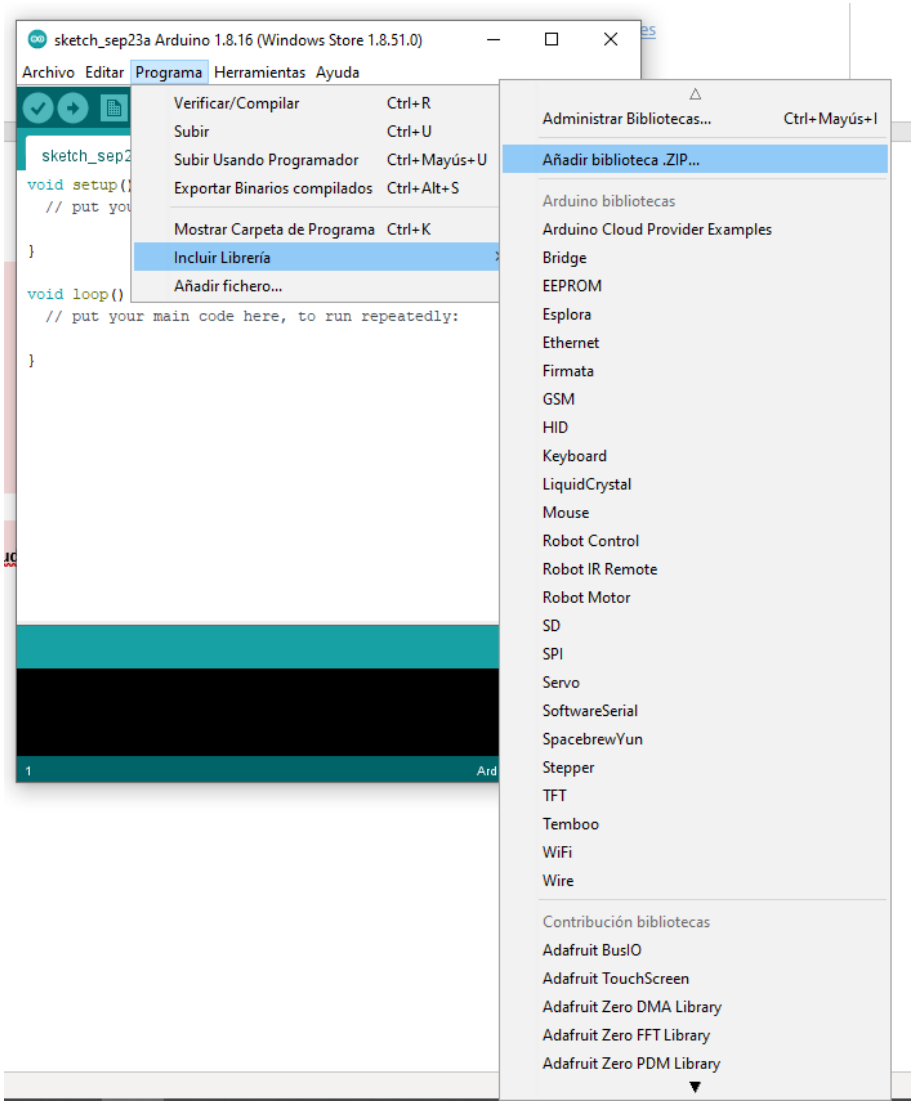


Ilustración 3 Añadir librería manualmente

A partir de este momento la librería estará disponible en la lista de librerías: **Programa -> Include Library -> KeyPad**

Si ya existiera os dará un error como muestra la Ilustración 4.

Ilustración 4 Error al cargar librería ya existente

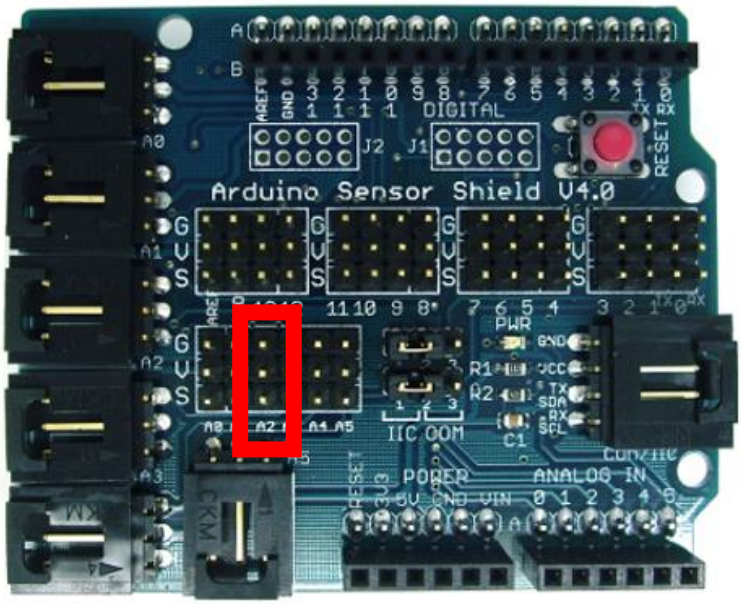
5. Ejemplos

Detección de luz

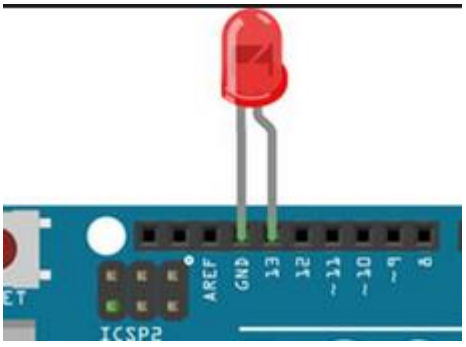
En este ejemplo vamos a utilizar un sensor de luz para monitorizar el nivel de luz de una habitación, en el caso de que el nivel de luz sea bajo encenderemos un LED.

Construcción del circuito:

1. **Montaje**
   a. **Opción 1**: protoboard + Arduino: conectamos el sensor de luz a: **GND (negro), 5V (Rojo) y A2 (azul).**
   b. **Opción 2**: sensor Shield. La forma de conexión más sencilla es utilizar una de las columnas de la placa de sensores.



2. Conectamos directamente sobre la placa un LED al Pin 13 y GND. El pin 13 incluye una resistencia, por lo que no hace falta colocar una. Realmente el pin 13 ya incluye un pequeño LED (justo debajo), pero conectamos nuestro LED para darle mayor visibilidad.



Programación de Arduino:

1. Inicializamos el puerto para registrar la salida en el método `setup()`. También activamos el pin digital 13 como salida.

2. En cada interacción del método `loop()` obtenemos el valor del sensor de luz leyendo directamente la entrada analógica.
3. Evaluamos el valor retornado por el sensor. En caso de que este valor sea bajo, encendemos el led.
4. Comprobamos el funcionamiento del programa: podemos tapar la fotorresistencia con la mano para obtener valores de luz bajos.

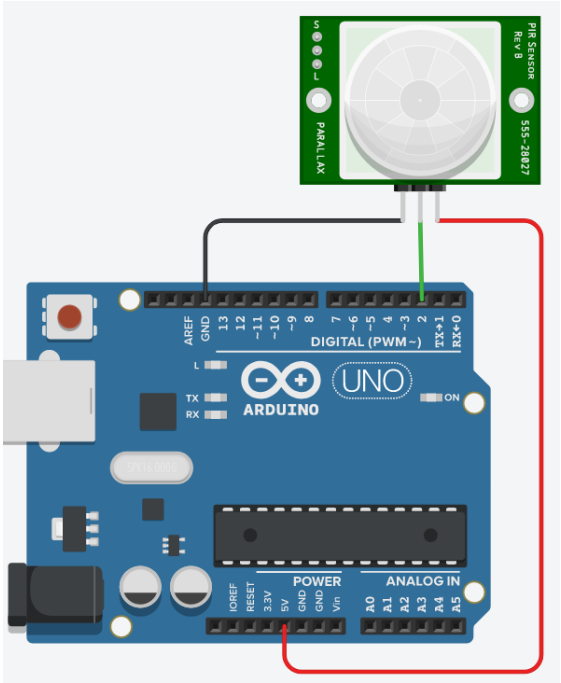
```
void setup(){
  //1.- No hace falta declarar las lecturas analógicas
  Serial.begin(9600);
  pinMode(13, OUTPUT);
}

void loop(){
  // Leer lectura analógica
  int lightValue = analogRead(A2);
  //2.-
  Serial.println("Valor: "+String(lightValue));

  //3.-
  if (lightValue < 200){
    digitalWrite(13, HIGH);
  } else {
    digitalWrite(13, LOW);
  }
}
```

Detección de movimiento

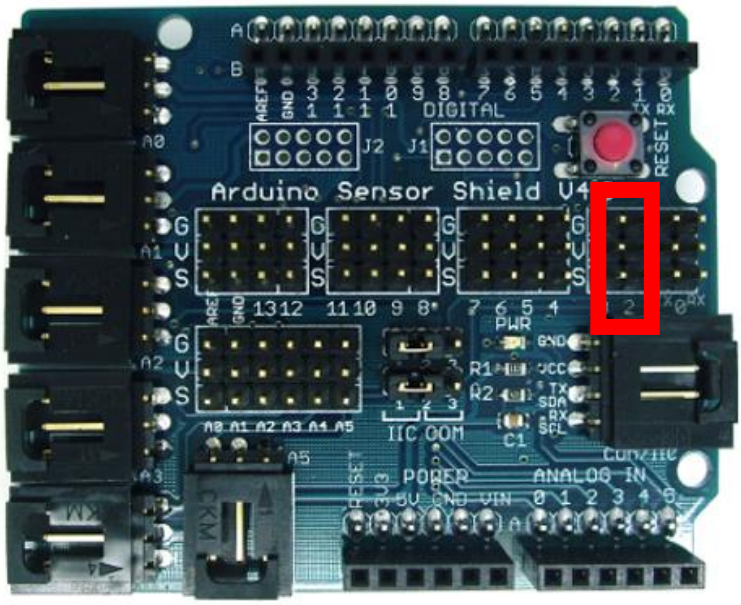
En este ejemplo utilizamos el sensor de movimiento de infrarrojos para detectar movimiento cerca del sensor.

Construcción del circuito:

1. Montaje
    a. **Opción** 1: conectamos el sensor PIR a: **GND, pin 2 y 5V**



b. **Opción 2**: sensor Shield. La forma de conexión más sencilla es utilizar una de las columnas de la placa de sensores:



2. Conectamos directamente sobre la placa un LED al Pin 13 y GND. El pin 13 incluye una resistencia, por lo que no hace falta colocar una.

Programación de Arduino:

1. Inicializamos el puerto para registrar la salida en el método `setup()`. También activamos el pin digital 2 como entrada e inicilizamos el puerto Serial para imprimir por consola.
2. En cada iteración del método `loop()` obtenemos el valor del sensor leyendo la entrada digital.

3. Evaluamos el valor retornado y hacemos una pequeña pausa entre mediciones. **Tarda en responder por consola y hay que ajustar la sensibilidad a la deseada, además de enfocarlo al sitio correcto, pues los reflejos o pequeños movimientos de la mesa harán que detecte movimiento.**
4. Código de ejemplo:

```
int pinSensor = 2;

void setup(){
  //1.-
  Serial.begin(9600);
  pinMode(pinSensor, INPUT);
}

void loop(){
  //2.-
  int pirValue = digitalRead(pinSensor);
  Serial.println("Valor: "+String(pirValue));

  //3.-
  if(pirValue == HIGH){
    Serial.println("Detectado movimiento");
  } else {
    Serial.println("- - - -");
  }

  delay(50);
}
```

Otros sensores digitales: sensor de vibración, infrarrojos, …

El funcionamiento del sensor de vibración es prácticamente igual que el del sensor PIR. Sin embargo, **las patillas del sensor cambian**, cuidado.

Control de distancia mínima

En este ejemplo utilizamos el sensor de ultrasonidos para detectar la distancia a la que colocamos un objeto.

Construcción del circuito:

1. Conectamos el sensor de ultrasonido a: **V5, Pin digital 9 (Trig), Pin digital 8 (Echo) y GND**

Programación de Arduino:

1. Creamos las variables globales que utilizaremos a lo largo de todo el programa: distance, responseTime, pinTrig, y pinEcho.
2. Inicializamos el puerto para registrar la salida en el método `setup()`. También activamos los 2 pines, uno como salida y otro como entrada, e inicilizamos el puerto Serial para imprimir por consola.
3. En cada interacción del método `loop()` colocamos el pin Trig a LOW y realizamos una breve pausa.
4. Emitimos la señal HIGH por el ping Trig y realizamos otra breve pausa.

5. Ya podemos leer el resultado del pin Echo. Para ello, utilizamos el método especial `pulseIn()`. Este método nos da los microsegundos que tarda el PIN en cambiar su estado HIGH / LOW.
    a. https://www.arduino.cc/en/Reference/pulseIn
6. Una vez tenemos los microsegundos que tarda el ultrasonido en impactar contra el objeto y volver, podemos calcular la distancia.
    a. La velocidad del sonido es de 343,2 m/s, que si lo transformamos en cm/s sería: 343,2 m/s * 100cm/m * 1s/1000000 ms = 0,03432 cm/ms. No obstante, como solo nos interesa lo que tarda en llegar, podemos dividir el tiempo de respuesta o la velocidad entre 2, pues sino estaríamos calculando lo que tarda en ir y volver. Esto nos da 0,01716 exactamente.
7. Finalmente añadimos una pequeña espera (quizá sea mejor probar con esperas más grandes).
8. Código de ejemplo:

```
    //1.-
    long distance;
    long responseTime;

    int pinTrig = 9;
    int pinEcho = 8;

    void setup(){
      //2.-
      Serial.begin(9600);
      pinMode(pinTrig, OUTPUT); /* Trig envía el pulso ultrasónico
*/
      pinMode(pinEcho, INPUT); /* Echo capta el rebote del pulso
ultrasónico*/
    }

    void loop(){
      //3.-
      digitalWrite(pinTrig, LOW); /* Por seguridad volvemos a poner
el Trig a LOW*/
      delayMicroseconds(5);

      //4.-
      digitalWrite(pinTrig, HIGH); /* Emitimos el pulso ultrasónico
*/
      delayMicroseconds(10);

      //5.-
      responseTime = pulseIn(pinEcho, HIGH); /* Medimos la longitud
del
      pulso entrante Cuanto tiempo tarda la entrada en pasar de HIGH
a LOW
      retorna microsegundos */
      Serial.println("Tiempo "+ String(responseTime)+"
microsegundos");

      //6.-
      distance = int(0.01716*responseTime); /* Calcular la distancia
      conociendo la velocidad */

      Serial.println("Distancia "+ String(distance)+"cm");
```

```
    //7.-
    delay(100);
}
```

Pantalla de segmentos

En este ejemplo vamos a utilizar una pantalla de segmentos de cuatro dígitos para imprimir un número por pantalla. Vamos a mostrar el valor de un entero en la pantalla.

Construcción del circuito:

1. Partiendo del circuito anterior conectamos la pantalla a: **CLK (~3), DIG (4), VCC (5V) y GND (GND).**

Programación de Arduino:

En primer lugar, para poder manejar la pantalla, necesitamos descargar e importar una nueva librería: **DigitalTube TM1637**: https://github.com/reeedstudio/libraries/tree/master/DigitalTube

Para agregar la librería, descargamos el archivo comprimido en zip y en el IDE pulsamos sobre **Programa -> Include Library -> Add. Zip Library**.

1. Al incluir la librería veremos que se genera un **#include** al principio de nuestro programa de la librería TM137.
2. Creamos las variables globales para los pines CLK y DIO. Declaramos un nuevo objeto de tipo TM1637 al cual le indicamos los pines a utilizar. La librería se encargará de su configuración.
3. En el método `setup()` inicializamos la pantalla y le establecemos el brillo. Posteriormente hay que realizar una pausa para darle tiempo a que se inicie. El método `init()` de TM1637 se encarga de iniciar todo lo necesario, incluido las salidas digitales. Con el método `set()` podremos establecer el brillo que deseemos entre 0 como el más oscuro y 7 el más claro.
4. En el interior del método `loop()`, una vez obtenida la distancia, la mostramos por pantalla. Para ello, usamos el método `display(índice de dígito, valor[0-9])`. Hay que tener en cuenta que debemos dividir la distancia en dígitos: unidades, decenas, centenas, millares. Para esto, tendremos que calcular en que posición va cada dígito antes de imprimirlo por la pantalla. Para ello, «jugaremos» con el tipo entero para quedarnos solo con la parte que nos importa del número y que el multiplicarlo de nuevo, nos quede solo esa unidad: 1,245 en un entero es 1, por 1.000 es 1.000, no 1245, y así sucesivamente.
5. ClearDisplay(): el método **clearDisplay** sirve para limpiar los datos enviados anteriormente a la pantalla de segmentos y así poder imprimir menos de 4 números en la pantalla, dejando alguno o varios sin utilizar, evitando que salga «basura» o los números previos en los que deseamos dejar en blanco.
6. Código de ejemplo:

```
//1.-
#include <TM1637.h>
```

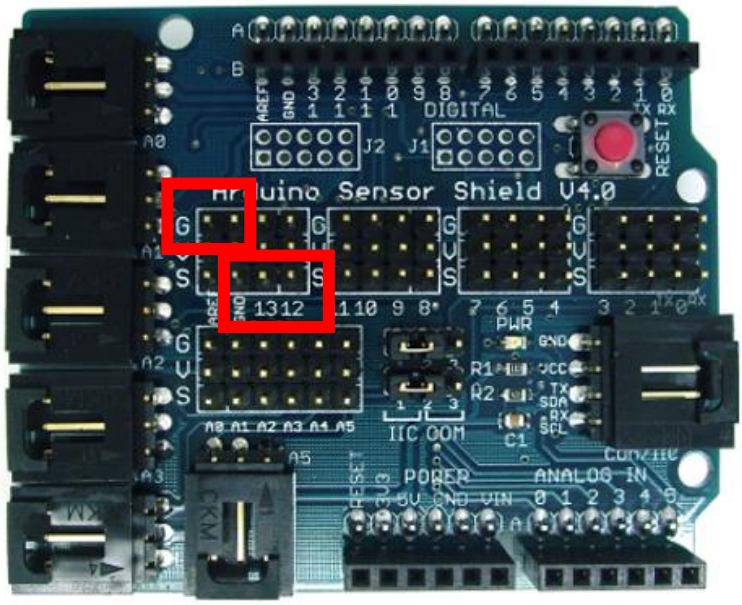
```
      long distance;

      //2.-
      int pinClk = 3;
      int pinDio = 4;

      TM1637 screen(pinClk, pinDio);

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

        //3.-
        screen.init();
        screen.set(BRIGHT_TYPICAL);
        //BRIGHT_TYPICAL = 2; BRIGHT_DARKEST = 0; BRIGHTEST = 7;

        delay(1500);// Esperamos a que se inicie la pantalla
      }

      void loop(){
        distance = 1245;

        //4.-
        int digit0 = distance/1000;
        Serial.println("Digit0 "+ String(digit0));

        int digit1 = (distance - digit0*1000)/100;
        Serial.println("Digit1 "+ String(digit1));

        int digit2 = (distance - (digit0*1000 + digit1*100))/10;
        Serial.println("Digit2 "+ String(digit2));

        int digit3 = distance - (digit0*1000 + digit1*100 +
digit2*10);
        Serial.println("Digit3 "+ String(digit3));

        screen.display(0, digit0);
        screen.display(1, digit1);
        screen.display(2, digit2);
        screen.display(3, digit3);

        delay(2000);

        //5.-
        screen.clearDisplay();
        screen.display(1, digit0);
        screen.display(2, digit1);

        delay(2000);
      }
```

LED RGB

En este ejemplo vamos a explorar el funcionamiento de un LED RGB.

Construcción del circuito:

1. Conectamos las patillas de LED utilizando el Shield de sensores a: **R** al **S9, G** al **S10 y B** al **S11, -** a cualquier **G.**



Programación de Arduino:

En el siguiente ejemplo controlamos los tres colores del LED con salidas digitales. Podemos colocar y establecer el color deseado en el LED RGB según la salida que pongamos a HIGH (enciende) o LOW (apaga).

```
int redPin = 9;
int greenPin = 10;
int bluePin = 11;

void setup() {
   pinMode(redPin, OUTPUT);
   pinMode(greenPin, OUTPUT);
   pinMode(bluePin, OUTPUT);
}

void loop() {
  digitalWrite(redPin, HIGH);
  digitalWrite(greenPin, LOW);
  digitalWrite(bluePin, LOW);
  delay(500);

  digitalWrite(redPin, LOW);
  digitalWrite(greenPin, HIGH);
  digitalWrite(bluePin, LOW);
  delay(500);

  digitalWrite(redPin, LOW);
  digitalWrite(greenPin, LOW);
  digitalWrite(bluePin, HIGH);
  delay(500);
}
```

En este otro ejemplo controlamos el LED con salidas digitales PWM, que nos permiten enviar señales analógicas y podemos establecer cada color de salida a un valor entre [0 – 255].

Esto nos proporciona el poder crear una gama más amplia de colores según las mezclas que realicemos.

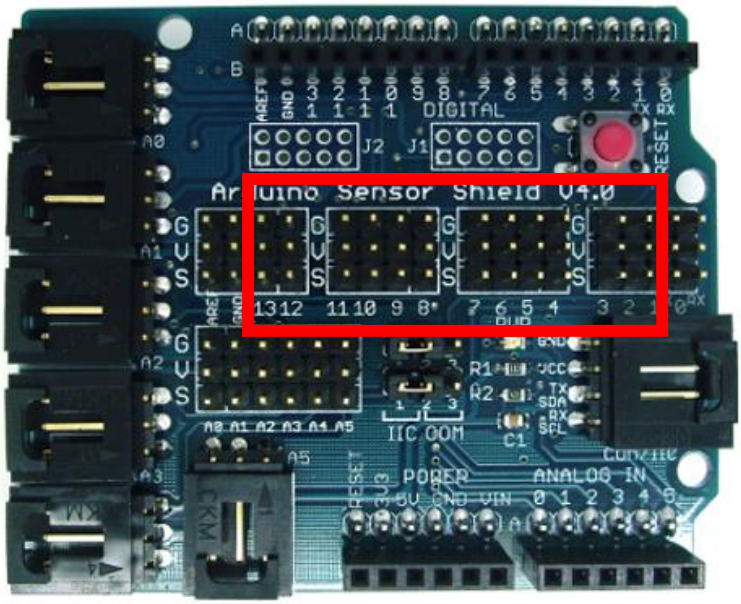
```
int redPin = 9;
int greenPin = 10;
int bluePin = 11;

int scaleColour = 0;

void setup() {
   pinMode(redPin, OUTPUT);
   pinMode(greenPin, OUTPUT);
   pinMode(bluePin, OUTPUT);
}

void loop() {
  analogWrite(redPin, 255 - scaleColour);
  analogWrite(greenPin, scaleColour);
  analogWrite(bluePin, scaleColour);
  delay(15);

  scaleColour++;
  if (scaleColour > 255)
    scaleColour = 0;
}
```

Servomotor 180°

En este ejemplo vamos a explorar el funcionamiento de un miniservomotor.

Construcción del circuito:

1. Conectamos el servomotor a: **V5 (rojo), Pin digital 8 (naranja) y GND (negro/marrón).**
2. La forma de conexión más sencilla es utilizar una de las columnas de la placa de sensores:



Programación de Arduino:

1. En primer lugar, importamos la librería que Arduino nos ofrece para controlar el servomotor. Después de importarla, observaremos que se ha incluido un nuevo #include en nuestro programa de la librería Servo.
2. Creamos la variable global de tipo Servo.
3. Inicializamos el puerto Serial para imprimir por consola y el puerto para registrar la salida en el método `setup()` y asociamos el pin 8 al servo. No hace falta especificar que el pin 8 será una salida digital, ya que se encarga la librería de hacerlo por nosotros.
4. En el método `loop()` movemos el servo a diferentes grados utilizando el método `write(grados)`. Después de cada movimiento realizamos una pausa.
5. Código de ejemplo:

```
//1.-
#include <Servo.h>

//2.-
Servo servo1;

void setup() {
 //3.-
 Serial.begin(9600);
 servo1.attach(8);
}

void loop() {
  //4.-
  servo1.write(180);
  Serial.println("180");
  delay(3000);
  servo1.write(90);
  Serial.println("90");
  delay(3000);
  servo1.write(0);
  Serial.println("0");
  delay(3000);
 }
```

No desmontéis el circuito, pues se utilizará en el siguiente ejemplo.

Teclado

En este ejemplo vamos a utilizar un teclado matricial para introducir el número de grados que debe girar un mini servomotor.

Construcción del circuito:

1. Partiendo del circuito anterior, el del servomotor, conectamos el teclado a: **2 - 5 y A0 - A3** de forma ordenada empezando por el conector de la izquierda.

Programación de Arduino:

Con el nuevo IDE; parece no ser necesaria incluirla. Si se trabaja con el IDE antiguo, tal vez hubiera que bajar, en algunos casos, la siguiente librería para poder manejar el teclado, necesitamos importar una nueva librería, la **KeyPad**:

https://cdn.shopify.com/s/files/1/0557/2945/files/KeyPad_bb5610fd-0c4c-4b0e-8c5e-1e5b3f83f47f.zip?4187195515480435262

Para agregar la librería descargamos el Zip y pulsamos sobre **Programa -> Include Library -> Add. Zip Library**

1. Continuamos modificando el programa anterior. Lo primero es incluir la librería del teclado matricial. Aquí veremos que se genera un **#include** al principio de nuestro programa de la librería Keypad (la K es mayúscula).
2. Creamos las variables globales para el número de filas y de columnas, y la matriz con todas las teclas que tiene nuestro teclado.
3. Creamos dos arrays indicando los pines que vamos a utilizar para las filas y las columnas.
4. Creamos un objeto de tipo Keypad, que recibe un KeyMap con las teclas, los pines de las filas, los pines de las columnas y el número de filas y columnas.
5. Finalmente, añadimos un buffer de lectura en el que iremos almacenando todas las teclas que pulsa el usuario.
6. En el método `loop()` pedimos que nos devuelva la tecla obtenida. Si el usuario no pulsa nada en cada iteración del bucle loop devolverá vacío ('\0'). Obtenemos la tecla pulsada con el método `getKey()` y comprobamos que no es vacía antes de continuar con el procesamiento. En caso de que no sea vacía la mostramos por consola.
7. Vamos a definir que cuando el usuario pulse la tecla '#' ya ha dejado de escribir. Cualquier tecla distinta a '#' la agregamos al buffer. Cuando el usuario pulsa '#' pasamos todo el contenido del buffer a un entero y lo vaciamos.
8. Utilizamos ese «valorNumerico» para indicarle al servomotor la posición que debe tomar.

```
//1.-
#include <Keypad.h>
#include <Servo.h>

Servo servo1;

//2.-
const byte nfilas = 4;
const byte ncolumnas = 4;
char teclas[nfilas][ncolumnas] = {
 {'1','2','3','A'},
 {'4','5','6','B'},
 {'7','8','9','C'},
 {'*','0','#','D'}
};

//3.-
byte pfilas[nfilas] = {2,3,4,5};  // Filas
byte pcolumnas[ncolumnas] = {A0,A1,A2,A3}; //Columnas

//4.-
Keypad teclado = Keypad(makeKeymap(teclas), pfilas, pcolumnas, nfilas, ncolumnas);

//5.-
String bufferLectura = "";
```

```
void setup(){
  Serial.begin (9600);
  servo1.attach(8);
  Serial.println("Setup()");
}

void loop(){
  //6.-
  char tecla = teclado.getKey();
  if (tecla != '\0'){
    Serial.println("tecla pulsada: "+String(tecla));

    if (isdigit(tecla)){
      bufferLectura = bufferLectura + tecla;
      Serial.println("buffer: "+bufferLectura);

//7.-
    } else if(tecla == '#'){
     // es # transformamos a entero
      int valorNumerico = bufferLectura.toInt();
      if (valorNumerico > 180){
        valorNumerico = 180;
      }
      Serial.println("Aceptado: "+String(valorNumerico));
      //8.-
      servo1.write(valorNumerico);
      bufferLectura = ""; // reinicio
    } else {
      Serial.println("¡Letras no!");
    }
}
```