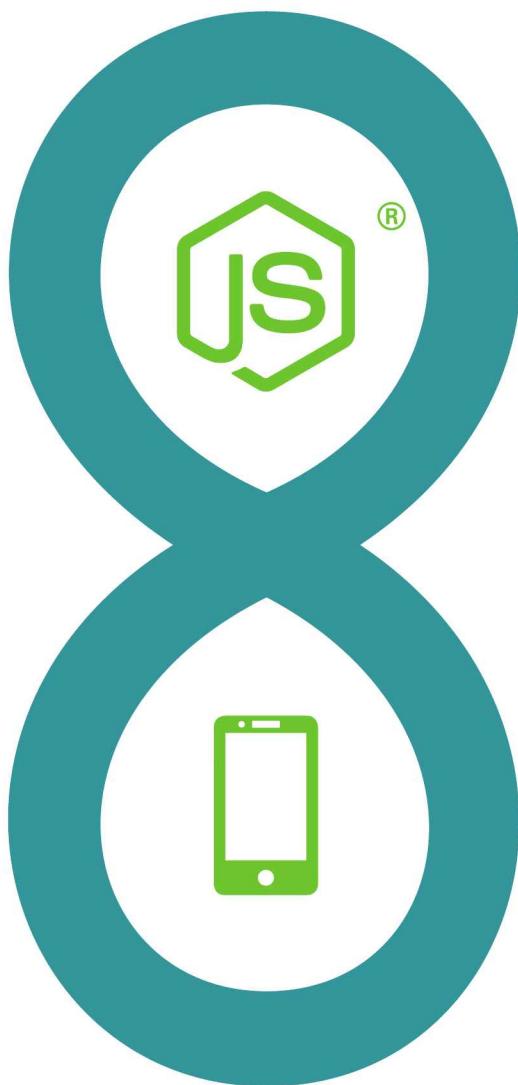


Arduino y Node.js



Aplicación móvil

Enciende la luz a un nuevo camino desde el clásico Hola Mundo al entorno App.

Guía práctica que une dos mundos apasionantes.
Por un lado el Hardware Libre con su representante Arduino y por otro el nuevo paradigma en desarrollos web Node.js

Javier Pardal

Gerente Inforede (inforede.es)

Moncho Pena

Director Técnico en Wentook (wentook.com)

Digitized by

Autores

Javier Pardal

Moncho Pena



Reconocimiento 4.0 Internacional (CC BY 4.0)

http://creativecommons.org/licenses/by/4.0/deed.es_ES

Diseño y maquetación

Pablo Villaverde

“640K deberían ser suficientes para todo el mundo”

Bill Gates, 1981

(Frase falsamente atribuida a Bill Gates)

Índice general

1 // Presentación	11
1.1 // Acerca de los autores	13
1.2 // ¿Qué vamos a aprender?	14
1.3 // Objetivos del libro	15
2 // Configurando el entorno de trabajo: "Hola mundo"	17
2.1 // Introducción	19
2.2 // Arduino	19
2.3 // Compilando Node JS	22
2.4 // Instalando Processing	24
2.5 // Hola mundo	24
2.5.1 // Hola mundo con Arduino	24
2.5.2 // Hola mundo en Node JS	25
2.5.3 // Hola mundo en Processing	26
2.6 // Anexo: Instalando el Android SDK	28
3 // Electrónica básica	29
3.1 // Introducción	31
3.2 // Voltaje	31
3.3 // Resistencia	31
3.4 // Ley de Ohm	31
3.5 // Potencia	32
3.6 // Símbolos y componentes	32
3.7 // Resistencias	33
3.7.1 // Asociación de Resistencias	34
3.8 // Condensador	36
3.9 // Placa de pruebas	36

Índice

4 // El Lenguaje de Arduino	39
4.1 // Introducción	41
4.2 // El hardware de Arduino	44
4.3 // Estructuras de control.	45
4.3.1 // Proposición if	45
4.3.2 // Bucle for	46
4.3.3 // Arrays	46
4.3.4 // While Loop	47
4.3.5 // Switch	48
4.4 // Leer desde Serial	49
5 // Ejemplos con Arduino	53
5.1 // Introducción	55
5.2 // Led parpadeante	55
5.2.2 // Material necesario.	55
5.2.3 // Conexionado	55
5.2.4 // Sketch	56
5.2.5 // Proyecto final	57
5.3 // Led desvaneciente	58
5.3.1 // Objetivo	58
5.3.2 // Material necesario.	58
5.3.3 // Conexionado	58
5.3.4 // Sketch	59
5.3.5 // Proyecto final	60
5.4 // Botón	60
5.4.1 // Objetivo	60
5.4.2 // Material necesario.	60
5.4.3 // Conexionado	61
5.4.4 // Sketch	62
5.4.5 // Proyecto final	63

5.5 // Fadding por potenciómetro.....	63
5.5.1 // Objetivo	63
5.5.2 // Material necesario.....	64
5.5.3 // Conexionado	64
5.5.4 // Sketch	65
5.5.5 // Proyecto final	66
6 // Ejemplos con Arduino Avanzados I	67
6.1 // Tono	65
6.1.1 // Objetivo	69
6.1.2 // Material necesario.....	69
6.1.3 // Conexionado	70
6.1.4 // Sketch	70
6.1.5 // Proyecto final	71
6.2 // Sensor de luz	72
6.2.1 // Objetivo	72
6.2.2 // Material necesario.....	72
6.2.3 // Conexionado	72
6.2.4 // Sketch	73
6.2.5 // Proyecto final	75
6.3 // Sensor de temperatura.....	75
6.3.1 // Objetivo	75
6.3.2 // Material necesario.....	75
6.3.3 // Conexionado	76
6.3.4 // Sketch	76
6.3.5 // Proyecto final	78
6.4 // Sensor de temperatura y humedad.....	78
6.4.1 // Objetivo	78
6.4.2 // Material necesario.....	78
6.4.3 // Conexionado	79
6.4.4 // Sketch	79
6.4.5 // Proyecto final	81

Índice

7 // Ejemplos con Arduino Avanzados II	83
7.1 // Servo	85
7.1.1 // Objetivo	85
7.1.2 // Material necesario	85
7.1.3 // Conexionado	86
7.1.4 // Sketch	86
7.1.5 // Proyecto final	87
7.2 // Sensor de movimiento	88
7.2.1 // Objetivo	88
7.2.2 // Material necesario	88
7.2.3 // Conexionado	88
7.2.4 // Sketch	88
7.2.5 // Proyecto final	89
7.3 // Mini estación meteorológica	90
7.3.1 // Objetivo	90
7.3.2 // Material necesario	90
7.3.3 // Conexionado	91
7.3.4 // Sketch	92
8 // Node JS y el Puerto Serial	95
8.1 // Introducción	97
8.2 // Express JS	97
8.3 // Sockets	101
8.4 // Node JS trabajando con Serial	104
9 // Aplicación móvil	111
9.1 // Introducción	113
9.2 // Angular JS	113
9.2.1 // Hola Mundo con Angular	114
9.2.2 // Chat con Angular JS y Express JS	116
9.3 // Hola mundo con Ionic Framework	121v

10 // Raspberry Pi	125
10.1 // Introducción	127
10.2 // Montando el hardware	127
10.3 // Instalando el Sistema Operativo	128
10.3.1 // Paso 1 (preparación de la tarjeta SD)	129
10.3.2 // Paso 2 (descarga de los archivos de instalación).	129
10.3.3 // Paso 3 (conectándolo todo)	129
10.3.4 // Paso 4 (instalación propiamente dicha)	129
10.4 // Control de los GPIO	133
11 // Ejercicios finales	137
11.1 // Introducción	139
11.2 // Enviando texto a un LCD	139
11.3 // Placa de Relés	146
11.4 // Moviendo un servo	151

Capítulo 1

Presentación

1.1 // Acerca de los autores

Javier Pardal

Es el gerente de una empresa especializada en Software Libre con más diez años de vida llamada Inforede. En su empresa ofrecen servicios informáticos: redes, administración de sistemas Linux, servidores LAMP, diseño de páginas web, videovigilancia, intranets ... En los últimos tiempos explora con éxito soluciones con Hardware Libre: en especial con Arduino. Por su personalidad y su experiencia dando cursos es un gran divulgador.



Moncho Pena

Actualmente es el CTO de codigo.co.uk una empresa con sede en Londres especializada en desarrollos en base web formada por un grupo interdisciplinar de varias nacionalidades. Durante los últimos años trabajó como Director Técnico para Chxpert en una multinacional francesa llamada Acticall. Se ha convertido en un especialista en Node JS y en sus interacciones con el Internet de las Cosas. Hace diez años cofundó Inforede, en esa etapa se dedicó a la programación en general. Participó activamente en la Asociación AGASOL y lideró la el proyecto HardProcessing junto a Wireless Galicia. Durante años fue formador en tecnologías donde fue conocido por su paciencia y perseverancia.



1.2 // Que vamos a aprender

Arduino será la base de nuestros experimentos por lo que tendremos que conocerlo a fondo. A través de ejercicios prácticos practicaremos todas las posibilidades de programación.

Un poco de electrónica básica. Seguro que alguna vez has estudiado algo de electrónica. Por ejemplo: seguro que sabes que cuando ponemos dos resistencias en paralelo el inverso de la resistencia equivalente es igual a los inversos de las resistencias sumados ¿O era al revés? ¿Esto no era para las resistencias en serie? Aclararemos estas dudas, simplemente para tener una referencia y no equivocarse. Los conocimientos adquiridos en el colegio están ahí tan solo hay que refrescarlos.

Para entender la programación de Arduino tenemos que conocer primero quién es su padre: Processing ¡No solo aprenderemos Arduino también conoceremos este increíble programa! Y por primera vez encenderemos una luz externa desde una aplicación en un PC.

Haremos tantos ejercicios con Arduino que nos sangrarán las yemas de los dedos de sacar y meter cables en la placa de prototipado. Soñaremos con resistencias y con leds, a lo mejor también con ovejas eléctricas.

Tendremos que ponernos al día con JavaScript. Para no reinventar la rueda se trabajará con Ionic Framework , que dispone de los accionadores básicos ya hechos, como son: interruptores, barras de deslizamiento, ...

Y si conocemos un poco de JavaScript hacer los programas de Node JS.

En el ejemplo práctico se verá cómo domotizar un invernadero, que al fin y al cabo no es más que una casa de plástico llena de plantas.

IMPORTANTE: Sólo vamos a usar Software Libre. Esto no es un limitante, ni mucho menos, es una gran ventaja. Por ello todos los ejemplos se harán con el sistema operativo GNU-Linux. En nuestro caso usaremos una Debian. Si no te gusta esta distribución, puedes usar cualquier otra, como Ubuntu, OpenSUSE, ... Todos los ejemplos de programación se pueden hacer en otros hábitats como Windows o Mac. Pero los caminos son diferentes. Si quieres sacarle todo el partido a un Arduino y usar una Raspberry Pi es un buen momento para aprender a usar Linux a nivel básico: instalar un programa, hacer un listado de archivos en una carpeta, ...

1.3 // Objetivos del libro

Se trata de marcar un camino. Por supuesto no es el único. Hay muchas más opciones: casi tantas como lenguajes de programación. De hecho hablaremos de alguna de ellas. Pero nuestra experiencia y sobre todo el ensayo - error nos han llevado hasta una solución que creemos óptima.

Vamos a empezar desde cero. Desde lo más básico hasta la prueba final donde se pondrá a prueba todos los conocimientos adquiridos. Se comienza con el “¡Hola Mundo!” clásico donde hacemos que un led parpadee con Arduino hasta encender una luz desde un móvil con conexión a internet en cualquier parte del mundo.

Al final poniendo en práctica todo el conocimiento montaremos las piezas del puzzle final y nos sentiremos orgullosos.

Capítulo 2

Configurando el

Entorno:

“Hola mundo”

2.1 // Introducción

Antes de empezar es necesario instalar las herramientas necesarias para poder desarrollar el curso.

Para trabajar con Arduino se utiliza su IDE (Integrated Development Environment).

Como al final del curso se usará Node JS se explica como compilarlo.

2.2 // Arduino

En este curso utilizamos la distribución de Linux GNU “Debian Wheezy”. Recomendamos descargar la máquina virtual de Tegnix:

https://aula.tegnix.com/files/coordinacion/aula_tegnix/Aula_Tegnix.ova

Recordar que el usuario es aula y la contraseña es tegnix.

Dentro de los recursos de este tema hay un vídeo de cómo instalarla.

De todos modos este manual también sirve para cualquier otra versión de GNU Linux basada en Debian como Ubuntu, Linux Mint, etc...

Antes de instalar cualquier programa en Debian debemos ver si el sistema está actualizado para ello

- Actualizamos las fuentes

```
apt-get update
```

- Descargamos e instalamos las últimas actualizaciones.

```
apt-get upgrade
```

Figura 2.1: Captura apt-get upgrade

Nota: al estar en un entorno de pruebas dentro en el terminal escribimos:

```
sudo -s
```

De esta forma se obtienen los privilegios de root y se puede instalar o compilar programas con todos los permisos necesarios. Aunque no sea un procedimiento ortodoxo, nos va a ahorrar tiempo.

Después de hacer el apt-get upgrade aparece una pantalla como esta:

```
root@tegnix: /home/aula # apt-get upgrade  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias  
Leyendo la información de estado... Hecho  
0 actualizados, 0 se instalarán, 0 para eliminar y 0 no actualizados.  
root@tegnix: /home/aula #
```

Figura 2.1: Captura apt-get upgrade

El comando llamado apt-cache sirve para hacer una búsqueda:

```
apt-cache search arduino
```

Aunque esto no es necesario, sirve para ir conociendo las herramientas esenciales de instalación de Debian e ir acostumbrándonos a usar el terminal. Veremos como resultado varios paquetes, algunos con el nombre de arduino y otros no.

Para instalar el IDE de Arduino simplemente escribimos:

```
apt-get install arduino.
```

Tardará un par de minutos ya que instala el OpenJDK que es la versión libre de la plataforma de desarrollo de Java.

Una vez instalado se puede acceder al programa desde el menú Aplicaciones > Electrónica > Arduino IDE.

En el sistema estamos logueados como un usuario sin todos los privilegios, no puede acceder al puerto USB donde conectamos el Arduino que será:

```
/dev/ttyUSB0
```

Al darle a aceptar e introducir la contraseña lo que está haciendo el sistema es añadir al Grupo tty (dueño de los puertos) y al grupo dialout al usuario aula. Escrito en comandos sería:

```
usermod -a -G tty aula
```

```
usermod -a -G dialout aula
```

Al conectar nuestro arduino y escribir el comando dmesg en el terminal se verá una línea hacia el final que pone:

Configurando el Entorno de trabajo

dmesg es un comando que lista el buffer de mensajes del núcleo.

Para ver que se ha conectado el USB también se puede hacer un listado

```
ls /dev/tty*
```

Y encontrar el arduino.

Ahora es el momento de abrir el programa. Si todo va bien se procede a seleccionar un par de Campos:

Desde **Herramientas > Tarjeta**, seleccionamos nuestro Arduino. Para este curso se recomienda usar el Arduino Uno, pero en el listado están todos los modelos actuales.

En **Herramientas > Puerto Serial**, sólo hay una opción. Es importante seleccionar el puerto USB sino el programa no funcionará.

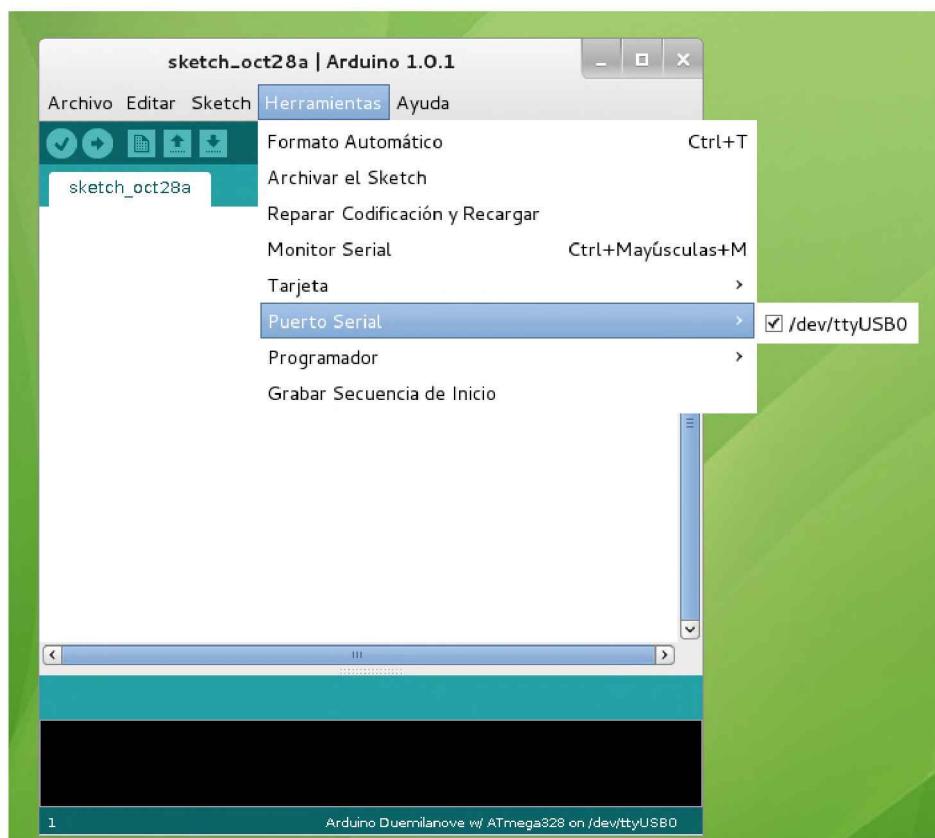


Figura 2.2: Captura seleccionar puerto USB

2.3 // Compilando Node JS

Para instalar Node JS se podrían usar los paquetes de Debian, pero en este caso se instala un programa muy anticuado, ya que los desarrolladores de Node JS sacan una nueva versión cada dos semanas. Por ello es importante aprender a compilar Node JS.

Lo primero será comprobar que están instaladas las herramientas básicas de compilación.

```
apt-get install build-essential
```

Se descarga el código fuente de su página oficial. Por ejemplo:

```
cd /usr/local/src
```

```
wget http://nodejs.org/dist/v0.10.28/node-v0.10.28.tar.gz
```

Se descomprime:

```
tar -xzvf node-v0.10.28.tar.gz
```

Entramos en el directorio:

```
cd node-v0.10.21
```

Se siguen los paso típicos para compilar un programa en Linux:

```
./configure
```

```
make
```

```
make install
```

Al terminar si todo ha ido bien

```
node -v
```

Con este comando se muestra la versión. También es importante comprobar que esté instalado npm (Node Packaged Modules) para ello:

```
npm -v
```

No es necesario obsesionarse con estar a la última pero al menos una vez cada dos meses deberíamos actualizar Node JS simplemente volviéndolo a compilar.

2.4 // Instalando Processing

En este curso se explica algún ejercicio con Processing para comprender mejor como es la programación en Arduino. Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital. Con este programa se pueden crear proyectos espectaculares que se conectan a un Arduino con muy pocas líneas de código.

Se entra en el directorio de nuestro usuario.

```
cd /home/aula
```

Se descarga el enlace desde su página web (recordemos que para la máquina virtual de Tegnix tendrá que ser la versión de 32 bits)

```
 wget http://download.processing.org/processing-2.1-linux32.tgz
```

Se descomprime

```
tar -xvzf processing-2.1-linux32.tgz
```

Se cambia al directorio que se acaba de crear

```
cd processing-2.0.3
```

Y se ejecuta el programa

```
./processing
```

Se verá una pantalla como esta



Figura 2.3: Captura Programa Processing

2.5 // Hola mundo

No se puede terminar el primer tema sin por lo menos hacer lo que se llama un “Hola mundo”, es decir, un programa que sirve para comprobar que todo lo que hemos instalado funciona y que es la expresión mínima de lo que vamos a hacer a continuación.

2.5.1 // Hola mundo con Arduino

En la placa del Arduino hay una zona donde están las salidas/entradas digitales.



Figura 2.4: Placa de Arduino

Se toma un led por la pata más corta (+ cátodo) y se introduce en el GND (Ground = Tierra), al mismo tiempo la pata más larga (- ánodo) irá en el PIN número 13. En ejemplos posteriores los ejemplos se harán sobre una placa de prototipado, pero en este ejemplo solo queremos probar que todo funciona.

Ejecutamos el Arduino IDE y abrimos uno de los ejemplos desde “**Archivo > Ejemplos > 01. Basics**” y se selecciona “Blink”. El programa que parece es similar a este:

```
int led = 13;

void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, HIGH);
  delay(1000);
}
```

En pocas palabras: se define la variable led con el valor 13, nada más arrancar el Arduino con este programa pondrá el led 13 a modo OUTPUT (salida), y repetirá eternamente poner a alta el led 13, esperar 1 segundo, poner a baja el led 13, esperar un segundo.

Configurando el Entorno de trabajo

Se carga este programa al Arduino pulsando el segundo botón que es un símbolo de flecha hacia la derecha.

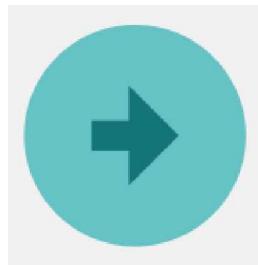


Figura 2.5: Botón upload IDE Arduino

¡Si ahora nos fijamos en el Arduino veremos parpadear el led!

2.5.2 // Hola mundo en Node JS

Se copia el ejemplo que trae la propia página de Node JS.

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('<h1>Hola mundo</h1>');
}).listen(1337, '127.0.0.1');

console.log('Server running at http://127.0.0.1:1337/');
```

En este programa se utiliza el método http y se crea un servidor que escucha en localhost en el puerto 1337, cuando un cliente accede por ejemplo desde un navegador va a recibir la cabecera de 200 (en pocas palabras que todo va bien) y para terminar muestra el texto “Hola mundo”.

Se guarda como server.js en /home/aula.

Se recomienda usar gEdit para realizar los programas. En caso de que no aparezca en el menú de **Aplicaciones > Accesorios** se procedería a instalarlo:

```
apt-get install gedit
```

Abrimos un terminal y ejecutamos el programa:

```
node server.js
```

El resultado:

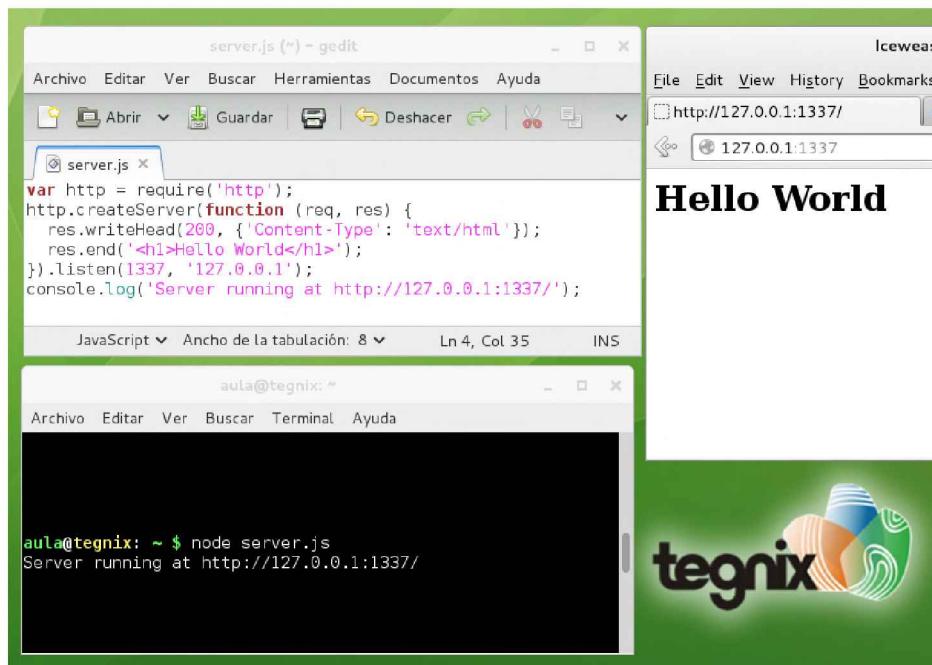


Figura 2.6: Node JS Ejemplo básico

2.5.3 // Hola mundo en Processing

Escribiendo tan solo una línea:

```
ellipse(50, 50, 80, 80);
```

Que significa: dibuja una elipse con centro de coordenadas x=50 e y=50. Con un alto y ancho de 80.

Al presionar el signo de Play de Processing se abrirá una nueva ventana con el dibujo de una elipse. El resultado:

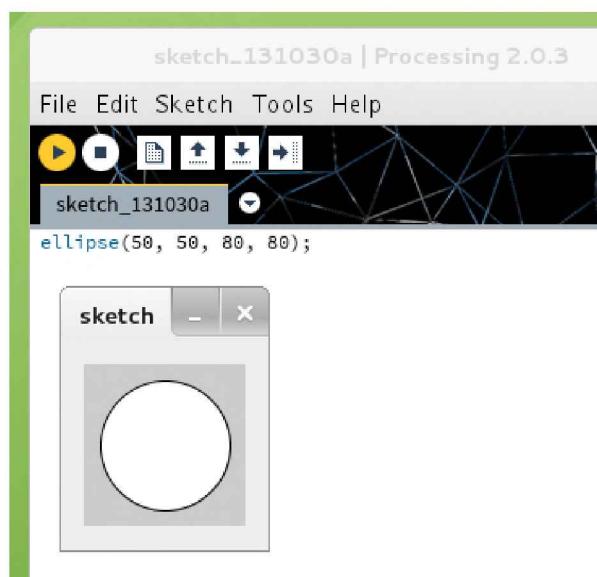


Figura 2.7: Processing ejemplo básico

Configurando el Entorno de trabajo

Un programa más complejo en Processing:

```
void setup() {  
    size(480, 120);  
}  
  
void loop() {  
    if (mousePressed) {  
        fill(0);  
    } else {  
        fill(255);  
    }  
    ellipse(mouseX, mouseY, 80, 80);  
}
```

Este programa consta de dos partes (¿A qué se parece al IDE de Arduino?). Primero un setup que sólo se ejecutará una vez al abrir el programa que define el tamaño de la ventana, y una función draw que siempre estará escuchando. El programa dibujará una elipse con el mismo alto y ancho (un círculo). Si tenemos el ratón presionado lo llenará de negro y si no será blanco.

El resultado:

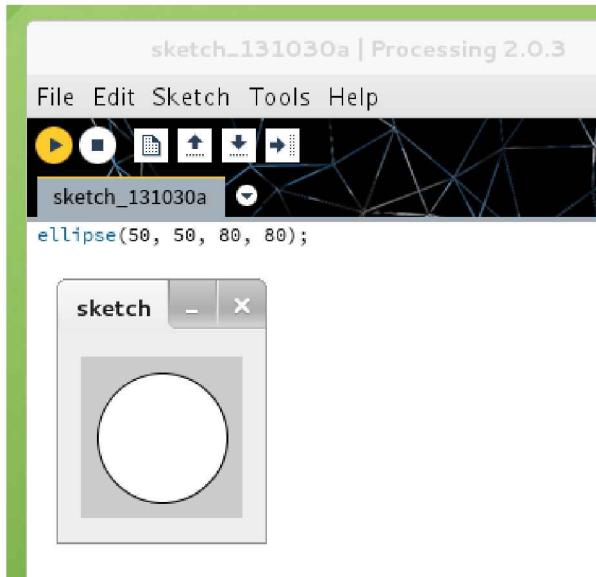


Figura 2.8: Processing movimiento

2.6 // Anexo: Instalando el Android SDK

Se necesita para poder probar nuestros ejemplos en un emulador o incluso en nuestros teléfonos móviles con este sistema operativo.

Desde la siguiente página se descarga el archivo para Linux “SDK Tools Only”:

<http://developer.android.com/sdk/index.html>

SDK Tools Only				
Platform	Package	Size	MD5 Checksum	
Windows 32 & 64-bit	android-sdk_r23.0.2-windows.zip	141435413 bytes	89f0576abf3f362a700767bdc2735c8a	
	installer_r23.0.2-windows.exe (Recommended)	93015376 bytes	7be4b9c230341e1fb57c0f84a8df3994	
Mac OS X 32 & 64-bit	android-sdk_r23.0.2-macosx.zip	90996733 bytes	322787b0e6c629d926c28690c79ac0d8	
Linux 32 & 64-bit	android-sdk_r23.0.2-linux.tgz	140827643 bytes	94a8c62086a7398cc0e73e1c8e65f71e	

Figura 2.9: Android SDK Descarga

Una vez descargado se descomprime:

```
tar -xzvf android-sdk_r23.0.2-linux.tgz
```

Se añade al “PATH” la carpeta de herramientas dentro del archivo .bashrc

```
PATH=$PATH:/home/aula/android-sdk-linux/tools/
```

Para que este archivo se recarge se ejecuta:

```
source /home/aula/.bashrc
```

Ahora al ejecutar en consola “android” se abre una ventana como esta:

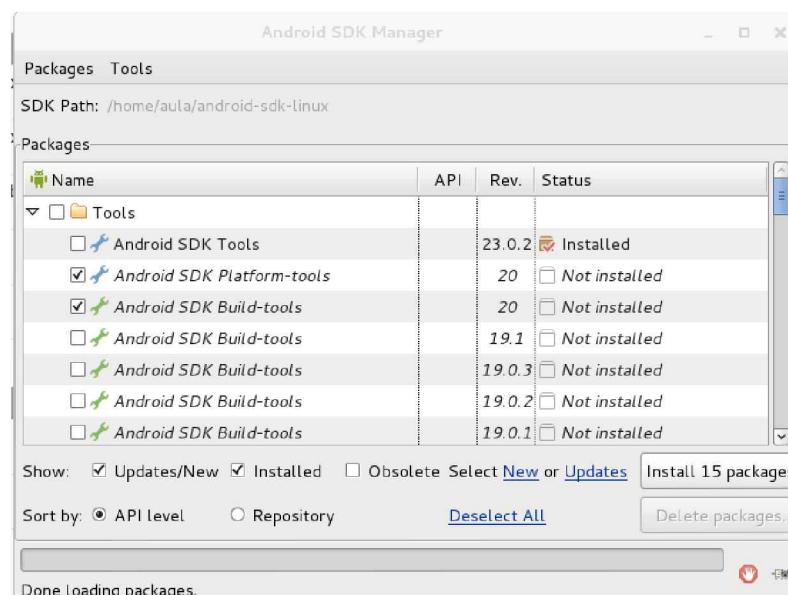


Figura 2.10: Android SDK ejecutado

Capítulo 2

Configurando el Entorno: “Hola mundo”

3.1 // Introducción

En este tema se repasan principios básicos de electrónica necesarios para comprender los circuitos de Arduino.

3.2 // Voltaje

La tensión eléctrica o diferencia de potencial es una magnitud física que cuantifica la diferencia de potencial eléctrico entre dos puntos. También se puede definir como el trabajo por unidad de carga ejercido por el campo eléctrico sobre una partícula cargada para moverla entre dos posiciones determinadas. Se puede medir con un voltímetro. Su unidad de medida es el voltio.

3.3 // Resistencia

Es la capacidad de oposición al paso de corriente que tiene un material. La unidad son Ohmios. Relaciona la corriente y el voltaje.

3.4 // Ley de Ohm

La ley de Ohm dice que la intensidad de la corriente que circula entre dos puntos de un circuito eléctrico es proporcional a la tensión eléctrica entre dichos puntos. Esta constante es la conductancia eléctrica, que es la inversa de la resistencia eléctrica.

$$I = V / R$$

La Intensidad se mide en Amperios.

3.5 // Potencia

Es la energía consumida por un componente electrónico. La unidad es el Vatio (Watts). Relaciona el voltaje y la corriente.

$$P = V * I$$

3.6 // Símbolos y componentes

Listado de símbolos más comunes en electrónica.

Símbolo	Nombre	Descripción
	Interruptor	El más sencillo de todos.
	LED	Muy importante el lado largo se llama Ánode (-) y el corto se llama Cátodo (+).
	Diodo	Al igual que los LED's sus terminales son ánodo y cátodo (este último, identificado con una banda en uno de sus lados), a diferencia de los LED's éstos no emiten luz.
	Resistencia	El código de colores se estudia más adelante en esta lección. Su unidad son los Ohmios y ofrecen resistencia a la corriente.
	Potenciómetros	Son resistencias variables, en su interior tienen una pista de carbón y un cursor que la recorre. Según la posición del cursor el valor de la resistencia de este componente cambiará.
	Foto-celda	También llamada LDR. Una foto-celda es un resistor sensible a la luz que incide en ella. A mayor luz menor resistencia, a menor luz mayor resistencia.
	Condensador	Almacenan energía, eso sí, se debe respetar la polaridad de sus terminales. El más corto es el negativo. o bien, podrás identificarlo por el signo en el cuerpo de componente.
	Transistores	Básicamente un transistor puede controlar una corriente muy grande a partir de una muy pequeña. muy común en los amplificadores de audio. Sus terminales son; Colector, Base y Emisor.
	Relé	Es un dispositivo de potencia, dispone de un electro-imán que actúa como intermediario para activar un interruptor, siendo este último totalmente independiente del electro-imán.

3.7 // Resistencias

El código de colores.

Color de la banda		Valor de la 1 ^a cifra significativa	Valor de la 2 ^a cifra significativa	Multiplicador	Tolerancia	Coeficiente de temperatura
Negro	[Negro]	0	0	1	-	-
Marrón	[Marrón]	1	1	10	1	100
Rojo	[Rojo]	2	2	100	2	50
Naranja	[Naranja]	3	3	1000	-	15
Amarillo	[Amarillo]	4	4	10000	4	25
Verde	[Verde]	5	5	100000	0,25	10
Morado	[Morado]	7	7	10000000	0,1	5
Gris	[Gris]	8	8	10^8	0,05	1
Blanco	[Blanco]	9	9	10^9	-	-
Dorado	[Dorado]	0	0	1	-	-
Plateado	[Plateado]	-	-	0,1	5	-
Ninguno		-	-	-	20	-

Por ejemplo:

- Primera línea - Azul: las decenas. 6
- Segunda línea - Verde: las unidades 5
- Tercera línea - Negro: el multiplicador 1

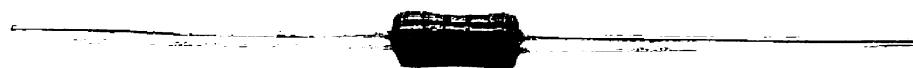


Figura 3.1: Resistencia

- Cuarta línea - Dorado: la tolerancia 2%
- Quinta línea - Rojo: 50ppm/°C

En resumen es una resistencia de 65Ω con una tolerancia del $\pm 2\%$.

3.7.1 // Asociación de Resistencias

Resistencia Equivalente (Figura c). Se denomina resistencia equivalente de una asociación respecto de dos puntos A y B, a aquella que conectada a la misma diferencia de potencial, U_{AB} , demanda la misma intensidad.

Asociación en Serie (Figura a). La resistencia equivalente a n resistencias montadas en serie es igual a la sumatoria de dichas resistencias.

Asociación en Paralelo (Figura b). La resistencia equivalente de una asociación en paralelo es igual a la inversa de la suma de las inversas de cada una de las resistencias.

Ejemplo:

Dadas 3 resistencias de 5Ω , 10Ω y 30Ω ¿Cuál sería su resistencia equivalente dispuestos en Serie? ¿Y en paralelo?

En serie es muy fácil:

$$R_E = 5 + 10 + 30 = 45\Omega$$

Y en paralelo

$$1/R_E = 1/5 + 1/10 + 1/30 = (6 + 3 + 1) / 30 = 10 / 30 = 1 / 3$$

Por lo que:

$$R_E = 3\Omega$$

Para tomar soltura con Node JS este es un ejemplo sencillo en Javascript que al pasarse las resistencias como argumento nos da el resultado de su Resistencia equivalente tanto en paralelo como en serial.

```
var myargs = process.argv.slice(2);
```

```
var re_serial=0;
var re_parallel=0;
var aux_parallel=0;
```

```
myargs.forEach(function(item) {
```

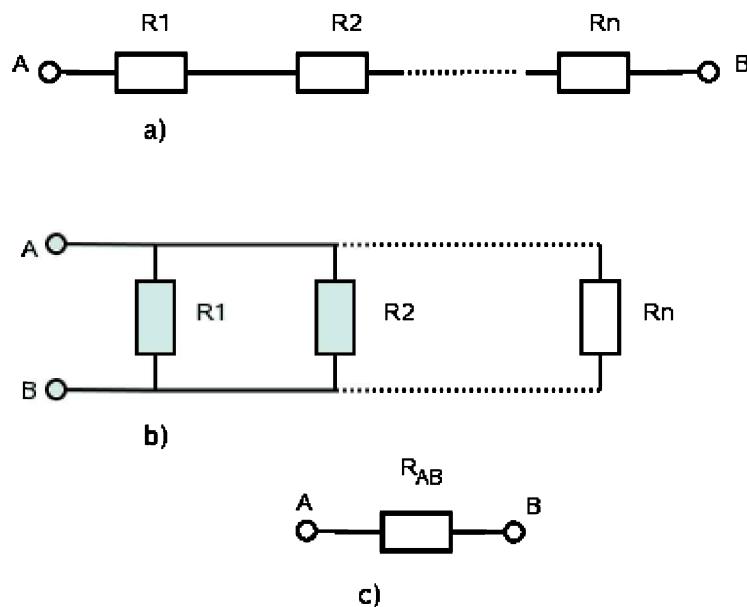


Figura 3.2: Asociación de resistencias

Electrónica Básica

```
re_serial=re_serial+parseFloat(item);
aux_parallel=aux_parallel+(1/parseFloat(item));
};

re_parallel=(1/aux_parallel);

console.log("re_serial: "+re_serial);
console.log("re_parallel: "+re_parallel);
```

Con esta variable recogemos los argumentos que le pasamos por consola.

```
var myargs = process.argv.slice(2);
```

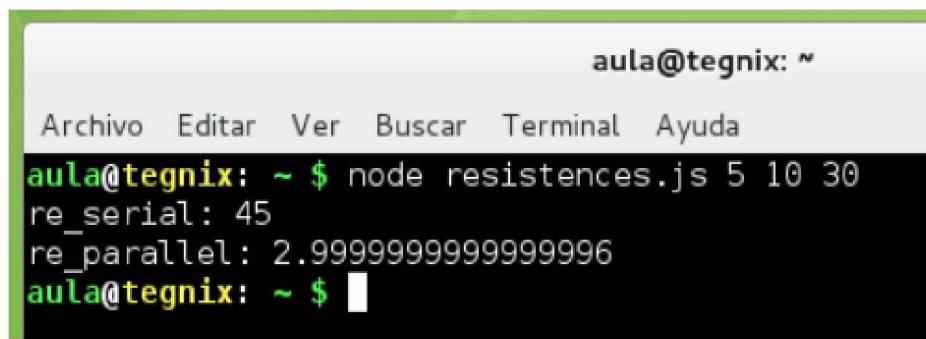
El dos significa que empezamos a recoger atributos a partir del tercer argumento.

Por ejemplo si se llama a este archivo resistence.js al ejecutar:

```
node resistence.js 5 10 30
```

```
Argumento 0: node
Argumento 1: resistence.js
Argumento 2: 5
...
...
```

En este caso se podría hacer la aproximación de 2.999999 a 3.



```
aula@tegnix: ~
Archivo Editar Ver Buscar Terminal Ayuda
aula@tegnix: ~ $ node resistences.js 5 10 30
re_serial: 45
re_parallel: 2.9999999999999996
aula@tegnix: ~ $
```

Figura 3.3: Resultado programa Node JS resistencias

3.8 // Condensador

Básicamente un condensador es un dispositivo capaz de almacenar energía en forma de campo eléctrico. Está formado por dos armaduras metálicas paralelas (generalmente de aluminio) separadas por un material dieléctrico.

La carga almacenada en una de las placas es proporcional a la diferencia de potencial entre esta placa y la otra, siendo la constante de proporcionalidad la llamada capacidad o capacitancia.

$$C=Q_1/(V_1-V_2)$$

C: Capacitancia o capacidad

Q1: Carga eléctrica almacenada en la placa 1.

V1-V2: Diferencia de potencial entre la placa 1 y la 2.

En el Sistema internacional de unidades se mide en Faradios (F), siendo 1 faradio la capacidad de un condensador en el que, sometidas sus armaduras a una d.d.p. de 1 voltio, estas adquieren una carga eléctrica de 1 culombio.

3.9 // Placa de pruebas

También conocida como placa de prototipado o en inglés “protoboard”, sirve para probar

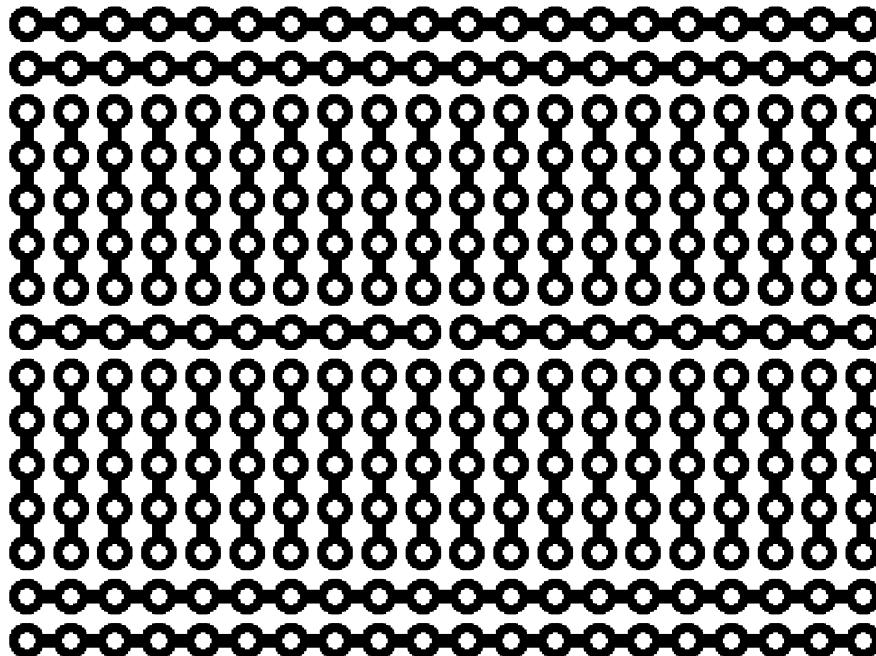


Figura 3.4: Esquema placa de prototipado

los circuitos antes de pasarlo a un modelo definitivo.

Este esquema sigue el patrón de líneas clásico de las placas de prueba.

Electrónica Básica

Con el programa Fritzing (<http://fritzing.org/>) se pueden hacer esquemas virtuales antes de pasar a la acción.

Para instalarlo:

```
apt-get install fritzing
```

Un ejemplo de esquema:

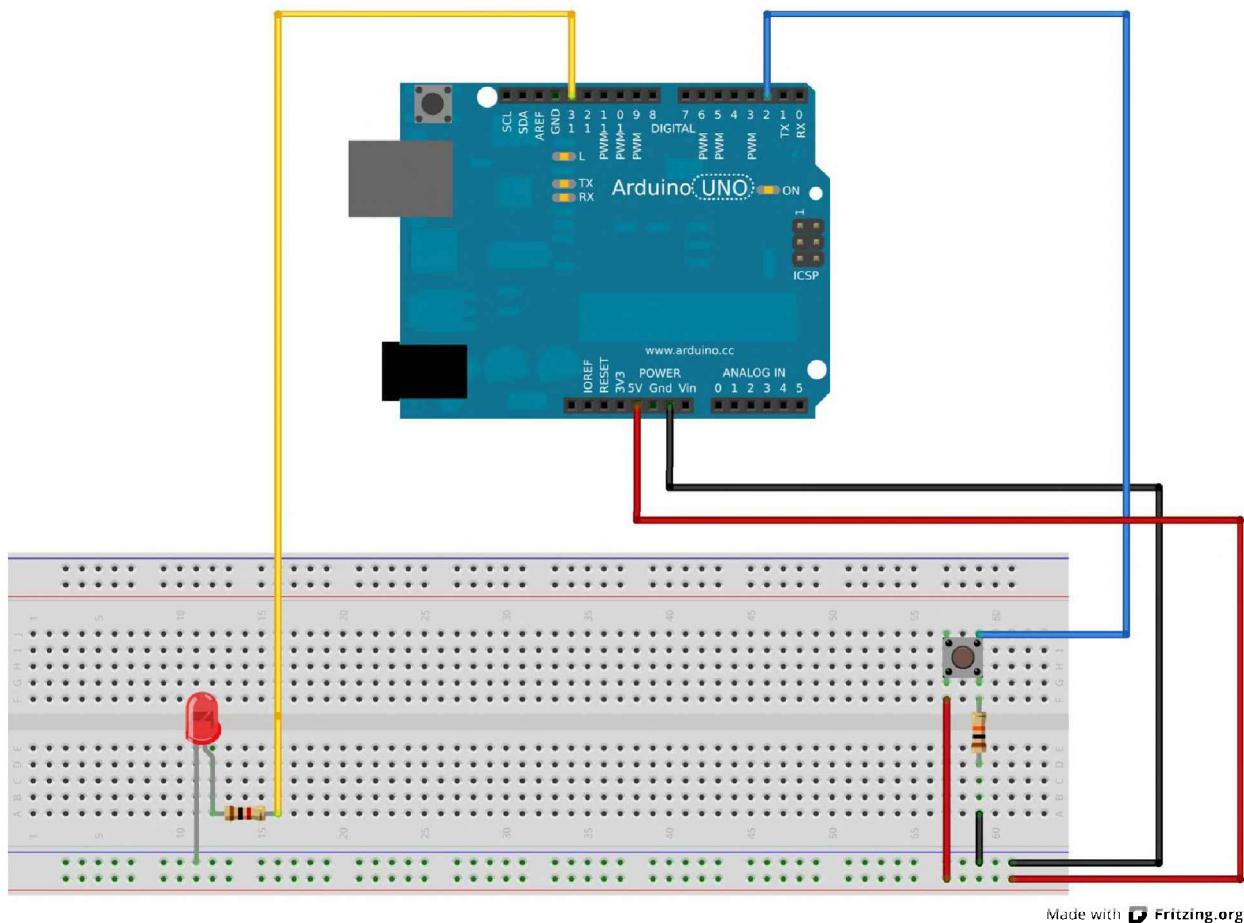


Figura 3.5: Ejemplo realizado con Fritzing

Capítulo 4

El Lenguaje

de Arduino

4.1 // Introducción

La plataforma Arduino se programa mediante el uso de un lenguaje propio basado en el lenguaje de programación de alto nivel Processing.

Por ello antes de empezar con el estudio del lenguaje de Arduino se realiza un ejemplo en Processing, para ir tomando soltura, ya que más adelante se realizará un sencillo Panel de Control con Processing.

Ejercicio donde se interactúa con teclas con el programa.



Figura 4.1: Processing Arduino

```
void setup() { //esta función se ejecuta tan solo una vez
    size(120, 120); //el tamaño de la pantalla
    smooth(); //redondea mejor las imágenes evita las “aristas”
}
```

```
void draw() { //esta parte siempre se está ejecutando
    background(204); //define el color de fondo en rgb
    if (keyPressed) { //controla si se presiona una tecla
        if ((key == 'h') || (key == 'H')) {
            line(30, 60, 90, 60);
        }
        if ((key == 'h') || (key == 'H')) {
            line(30, 20, 90, 100);
        }
        ...
        line(30, 20, 30, 100);
        line(30, 20, 30, 100);
    }
}
```

Al ejecutar el programa:

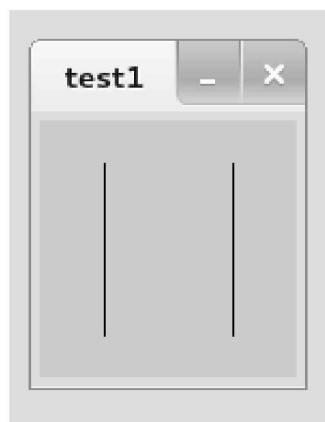


Figura 4.2: Processing ejecutado

El Lenguaje de Arduino

Una de las ventajas de Processing es que se pueden generar programas compilados listos para su ejecución. Para ello vamos a Archivo y Exportar Aplicación. Si queremos hacer un ejecutable para Linux



Figura 4.3: Processing exportando aplicación

Una vez terminado el proceso accediendo a la carpeta personal en "Sketchbook" al entrar en la carpeta con el nombre del programa verá la aplicación que se puede directamente ejecutar.

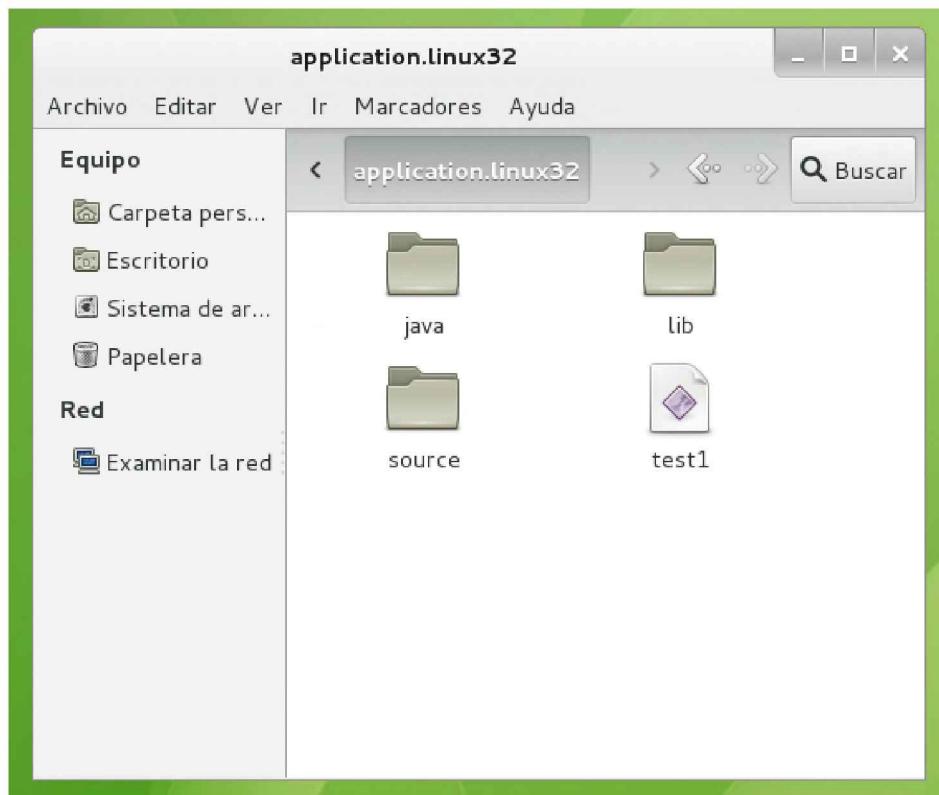
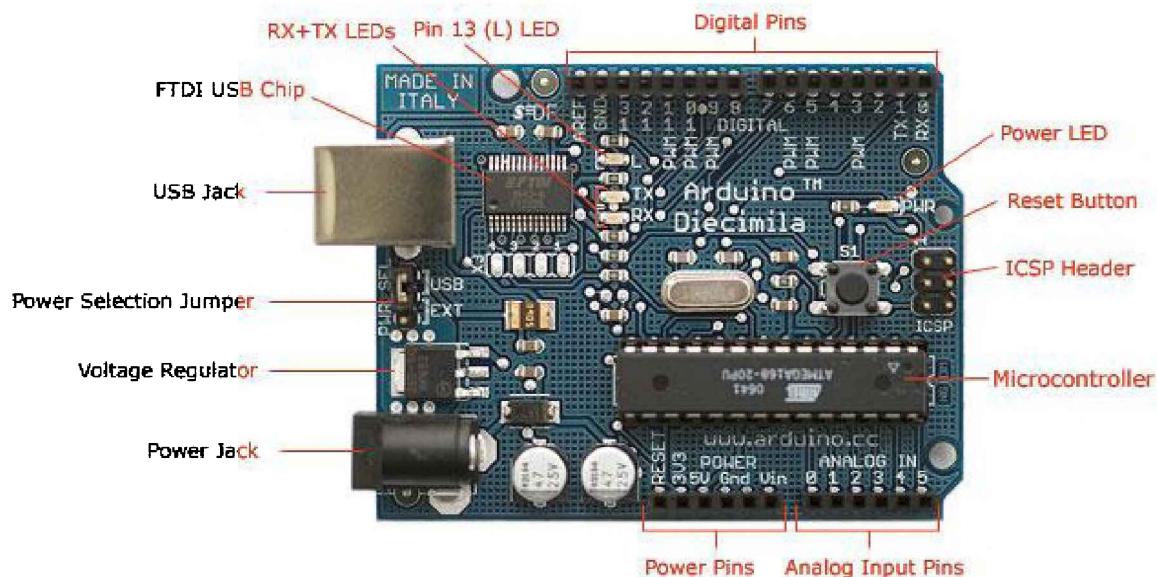


Figura 4.4: Processing el ejecutable

4.2 // El hardware de Arduino

A partir de ahora para entender los programas que se van a explicar a continuación se necesita comprender el hardware de Arduino.

Esto es una placa de Arduino:



Photograph by SparkFun Electronics. Used under the Creative Commons Attribution Share-Alike 3.0 license.

Figura 4.5: Placa de Arduino

Partes importantes

- **Conector de Potencia:** para alimentar el Arduino, se usa cuando no está conectado a un cable USB.
- **Conector USB:** la forma más sencilla de subir programas.
- **Pins digitales:** pueden ser de Entrada o Salida y su valor es de 1 o 0.
- **Microcontrolador:** el cerebro del Arduino, es donde se alojan los programas.
- **Pins analógicos:** su valor varía entre 0 y 255.
- **Pins de potencia:** hay dos uno de 5V y otro de 3.3V sirven para alimentar las diferentes partes de los proyectos como un LED o un Emisor de Radio Frecuencia.

4.3 // Estructuras de control

4.3.1 // Proposición if

La declaración If es la estructura de control más básica de todas.

```
if (someCondition) {  
    // hace cosas si la condición es verdad  
}
```

Existe una variación llamada if-else que se escribe:

```
if (someCondition) {  
    // hace cosas si la condición es cierta  
} else {  
    // hace cosas si la condición es falsa  
}
```

Y por último else-if, en la que se comprueba una segunda condición en caso de que la primera sea falsa:

```
if (someCondition) {  
    // hace cosas si la condición es cierta  
} else if (anotherCondition) {  
    // hace cosas sólo si la primera condición es falsa  
    // y la segunda es cierta  
}
```

4.3.2 // Bucle for

La estructura “for” es muy útil para la mayoría de las operaciones repetitivas, y habitualmente se usa para operaciones con vectores. Por ejemplo

```
for (int i=0; i <= 255; i++){  
    //haz cosas  
    ...  
}
```

Consta de tres partes

- Se inicializa int i=0
- Cada vez que se pasa por el bucle se comprueba si se cumple la condición, en este caso que i es menor o igual que 255
- Por último se incrementa el valor que en este caso es sumar uno i++

4.3.3 // Arrays

Esta variación del ejemplo For Loop muestra como usar un array. Un array o matriz es una variable con múltiples partes. Los arrays son muy útiles en programación para almacenar y mostrar datos.

```
int timer = 100;  
  
int ledPins[] = {  
    2, 7, 4, 6, 5, 3 };           // Esto es un array  
int pinCount = 6;                // El número de pins o de elementos del array  
  
  
void setup() {  
    // Se inician todos los pines a alta  
    for (int thisPin = 0; thisPin < pinCount; thisPin++) {  
        pinMode(ledPins[thisPin], OUTPUT);  
        ...  
    }  
  
  
    void loop() {  
        // bucle del más pequeño al más alto
```

El Lenguaje de Arduino

```
for (int thisPin = 0; thisPin < pinCount; thisPin++) {  
    // turn the pin on:  
    digitalWrite(ledPins[thisPin], HIGH);  
    delay(timer);  
    // turn the pin off:  
    digitalWrite(ledPins[thisPin], LOW);  
}  
  
// bucle de mayor a menor  
for (int thisPin = pinCount - 1; thisPin >= 0; thisPin--) {  
    // turn the pin on:  
    digitalWrite(ledPins[thisPin], HIGH);  
    delay(timer);  
    // turn the pin off:  
    digitalWrite(ledPins[thisPin], LOW);  
}  
}
```

4.3.4 // While Loop

A veces es necesario que todo el programa se pare hasta que una condición dada sea verdadera. Esto es muy útil por ejemplo para escuchar el puerto serial. Por ejemplo

```
while (digitalRead(2) == HIGH) { // mientras un pulsador conectado a la salida digital 2 este pulsado haz ...  
    //haz cosas  
}
```

4.3.5 // Switch

Una estructura ‘if’ te permite decidir entre dos opciones , verdadero o falso. Cuando existe más de una opción, puedes usar muchos if o puedes usar la estructura switch. La estructura switch te permite elegir entre muchas opciones discretas. Este tipo de control de control es muy importante para trabajar con serial.

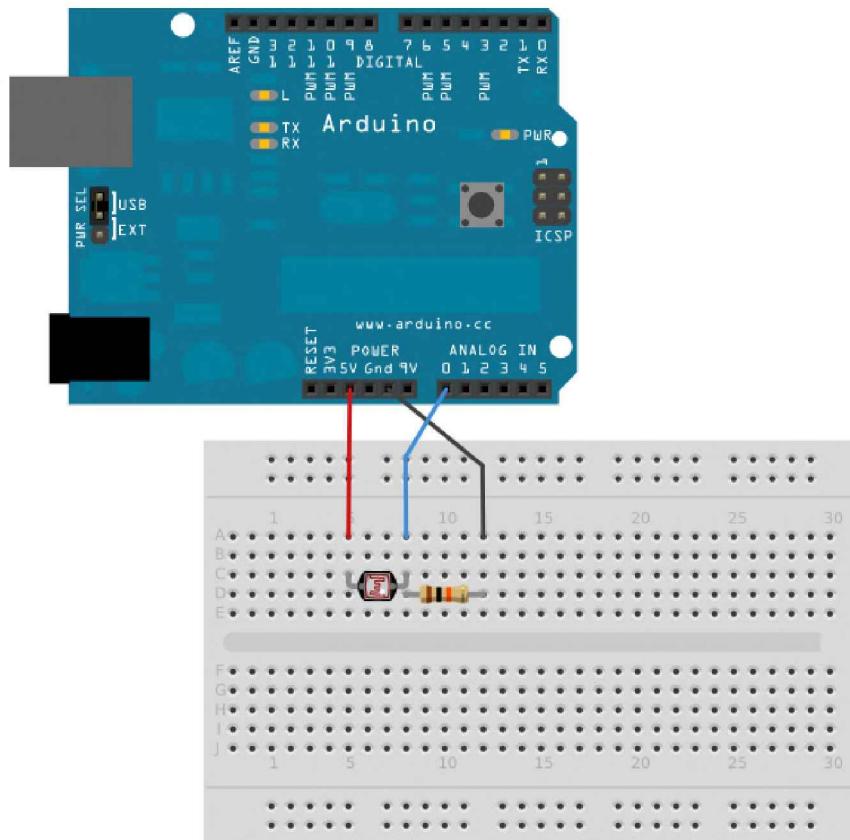


Figura 4.6: Ejemplo sensor de luz con Arduino

En este caso tenemos un sensor de luz que devuelve un valor entre 0 y 600, cuanto más alto es el valor más luminosidad.

```
const int sensorMin = 0;
const int sensorMax = 600;
void setup() {
    // Se inicia la comunicación serial
    Serial.begin(9600);
}

void loop() {
    // leemos el sensor
    int sensorReading = analogRead(A0);
```

```
int range = map(sensorReading, sensorMin, sensorMax, 0, 3);

// Haz algo diferente según el rango de respuesta

switch (range) {

    case 0: // la mano está sobre el sensor
        Serial.println("dark");
        break;

    case 1: // la mano está cerca del sensor
        Serial.println("dim");
        break;

    case 2: // la mano está a unos centímetros del sensor
        Serial.println("medium");
        break;

    case 2: // no hay nada frente al sensor
        Serial.println("medium");
        break;

}

delay(1000);

// se escribe un delay para darle estabilidad a la aplicación

}
```

4.4 // Leer desde Serial

Para comunicarse con Arduino a parte de poder hacerlo mediante botones o sensores, un método muy importante es mediante serial (o USB).

De esta forma se puede enviar órdenes directas.

La lectura de Serial en Arduino funciona de la siguiente forma: al recibir una cadena cualquiera la va leyendo carácter a carácter. Por lo tanto debemos colocar un límite para quedarnos con ciertas cadenas o valores.

A tener en cuenta: no se pueden mezclar en un Serial.print diferentes tipos de variables.

Este es un ejemplo completo. Se sugiere que antes de seguir con el curso se comprenda y se realicen pruebas hasta dominar esta técnica.

```
//Ejemplo comunicación Serial

String content = ""; // la variable donde almacenamos el contenido
char character; // el carácter que vamos leyendo
int number; // variable número entero
int myInts[6]; // un array de enteros de dimensión 6
int count=0; // un contador
int val; // el valor final

void setup() {
    // Inicializamos el serial
    Serial.begin(9600);
    Serial.println("#####");
    Serial.println("Funcion MAP");
    Serial.println("#####");
    Serial.println("");
    Serial.println("Por definicion:");
    Serial.println("map (value, fromLow, fromHigh, toLow, toHigh)");
    Serial.println("");
    Serial.println("Introduzca los valores según la función map, la orden que finaliza es *");
    Serial.println("");
}

void loop() {
}

if (Serial.available()) {

    character = Serial.read(); // recordar que se lee carácter a carácter
```

El Lenguaje de Arduino

```
if (character == '*') {  
    // importante: es el limitador cuando nos encontramos con esto hacemos cosas.  
  
    number=content.toInt();  
  
    //recibimos una cadena por lo tanto hay que pasarlo a entero  
    myInts[count]=number; // introducimos el primer número  
    count++;  
  
    switch (count) { //según el valor se imprime una frase u otra  
  
        case 1:  
            Serial.print("val: ");  
            Serial.println(content);  
            break;  
  
        case 1:  
            Serial.print("fromLow: ");  
            Serial.println(content);  
            break;  
  
        case 1:  
            Serial.print("fromHigh: ");  
            Serial.println(content);  
            break;  
  
        case 1:  
            Serial.print("toLow: ");  
            Serial.println(content);  
            break;  
    }  
}
```

```
case 5:  
//este es el último valor que se necesita, por lo tanto se procede a imprimir el resultado  
    Serial.print("toHigh: ");  
    Serial.println(content);  
    val = map(myInts[0], myInts[1], myInts[2], myInts[3], myInts[4]);  
    Serial.print("Y este es el resultado:");  
    Serial.println(val);  
    count=0; // se vuelve a poner el contador a cero  
    break;  
}  
content=""; // y el contenido también se vacía  
} else {  
    content.concat(character);  
// si no recibimos el valor de control usamos esta función para concatenar los caracteres  
}  
}  
}  
}
```

Capítulo 5

Ejemplos con

Arduino

5.1 // Introducción

Mediante estos ejemplos básicos, aprenderemos a utilizar las funciones que nos proporciona el IDE de Arduino, además de ir cogiendo experiencia en el montaje de los circuitos.

5.2 // Led parpadeante

El primer ejercicio con Arduino, simplemente consiste en encender y apagar un led a intervalos de 1 segundo.

La función digitalWrite(), escribe un valor HIGH ó LOW hacia un pin digital.

5.2.2 // Material necesario:

- Arduino.
- 1 Led.
- 1 Resistencia 1k ohm.
- Placa de prototipados.
- Cables de conexiones.



Figura 5.1: Leds

5.2.3 // Conexionado:

Conectamos el pin 13 a un extremo de la resistencia y el otro extremo al ánodo (positivo) del diodo led (es el extremo más largo). El cátodo lo conectamos a masa (gnd).

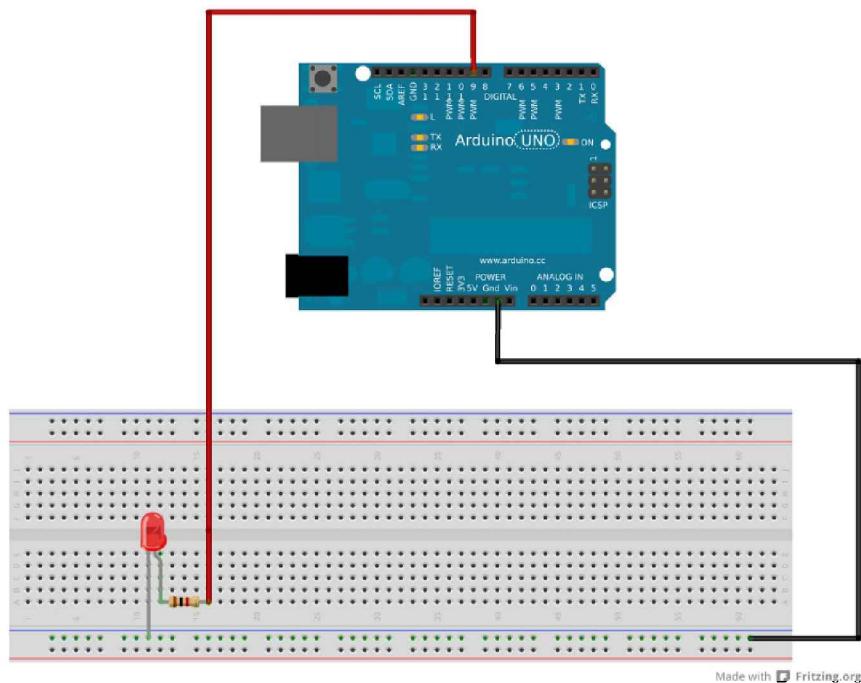


Figura 5.2: Esquema LED parpadeante

5.2.4 // Sketch:

```
//Sketch para hacer parpadear un led.
```

```
int buttonPin = 2;      // Pin donde conectamos el switch.  
int ledPin = 13;        // Pin donde conectamos el led.  
  
int buttonState = 0;    // variable para almacenar el estado del boton.  
  
void setup() {  
    pinMode(ledPin, OUTPUT); //Inicializamos el led como salida.  
    pinMode(buttonPin, INPUT); //Inicializamos el boton como entrada.  
}  
  
void loop() {  
    // leemos el estado del pin digital 2 y lo almacenamos  
    // en la variable.  
    buttonState = digitalRead(buttonPin);  
}
```

Ejemplos con Arduino

```
//Comprobamos si el boton esta pulsado.  
if (buttonState == HIGH) {  
    // Si es verdadero, encendemos el led.  
    digitalWrite(ledPin, HIGH);  
}  
  
else {  
    // de lo contrario, lo apagamos.  
    digitalWrite(ledPin, LOW);  
}  
}
```

5.2.5 // Proyecto final:

Así quedaría nuestro primer proyecto una vez hechas todas las conexiones.

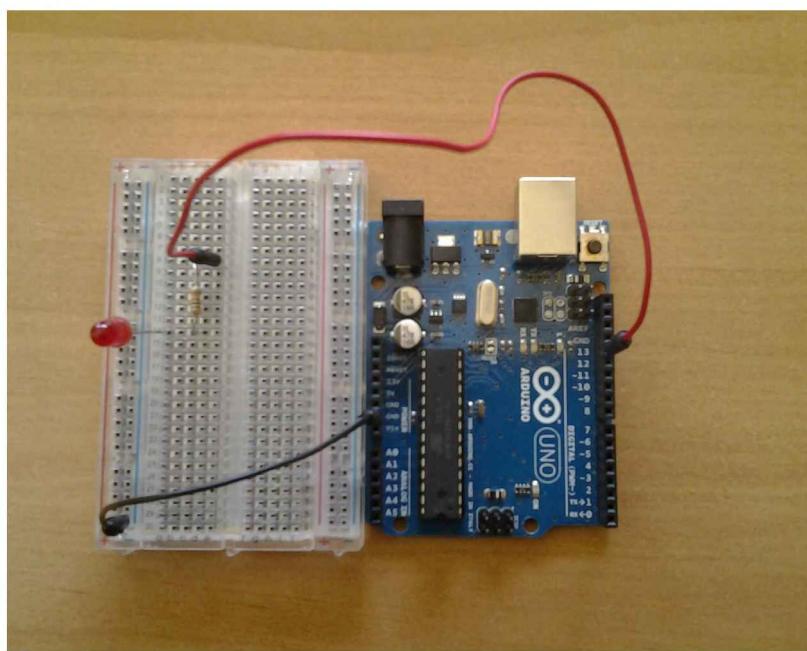


Figura 5.3: Proyecto LED final

5.3 // Led desvaneciente

5.3.1 // Objetivo:

Encender y apagar un led de forma evanescente.

La función `analogWrite()`, escribe un valor analógico (PWM) en un pin. Puede ser usado para controlar la luminosidad de un LED o la velocidad de un motor.

En la mayoría de las placas Arduino (aquellas con el ATmega168 o ATmega328), se podrá generar señales PWM en los pines 3, 5, 6, 9, 10, y 11.

5.3.2 // Material necesario:

- Placa Arduino.
- 1 Led.
- 1 Resistencia 1k ohm.
- Placa de prototipados.
- Cables de conexión.

5.3.3 // Conexionado:

Conectamos el pin 9 a un extremo de la resistencia, y el otro extremo de la misma al ánodo (positivo) del diodo led, (es el extremo más largo). El cátodo lo conectamos a masa (gnd).

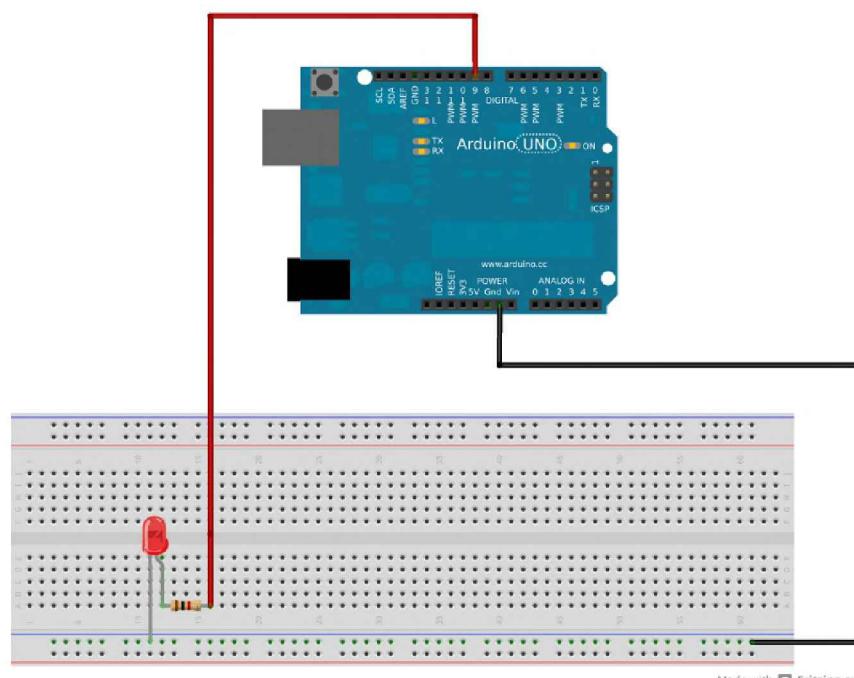


Figura 5.4: Proyecto LED desvaneciente

5.3.4 // Sketch:

```
/*
Ejemplo de uso de la funcion analogWrite() para conseguir
un efecto “fade” en un led conectado al pin 9.

*/
int led = 9;           // pin al que conectamos el led.
int brightness = 0;    // brillo del led.
int fadeAmount = 5;    // numero de pasos para el efecto fade.

// Funcion que solo se ejecutara 1 vez:
void setup() {
  pinMode(led, OUTPUT); //declaramos el pin 9 como salida.

}

// funcion que se ejecutara en bucle una y otra vez:
void loop() {
}

analogWrite(led, brightness); //Establece el brillo del pin 9.

//Para cada bucle el brillo se incrementara en pasos de 5 .
brightness = brightness + fadeAmount;

//Invierte la direccion del efecto fading al llegar al final:
if (brightness == 0 || brightness == 255) {
  fadeAmount = -fadeAmount ;
}

// Esperamos 30 milisegundos para apreciar el efecto:
delay(30);

}
```

5.3.5 // Proyecto final:

Aspecto final del prototipo una vez montado:

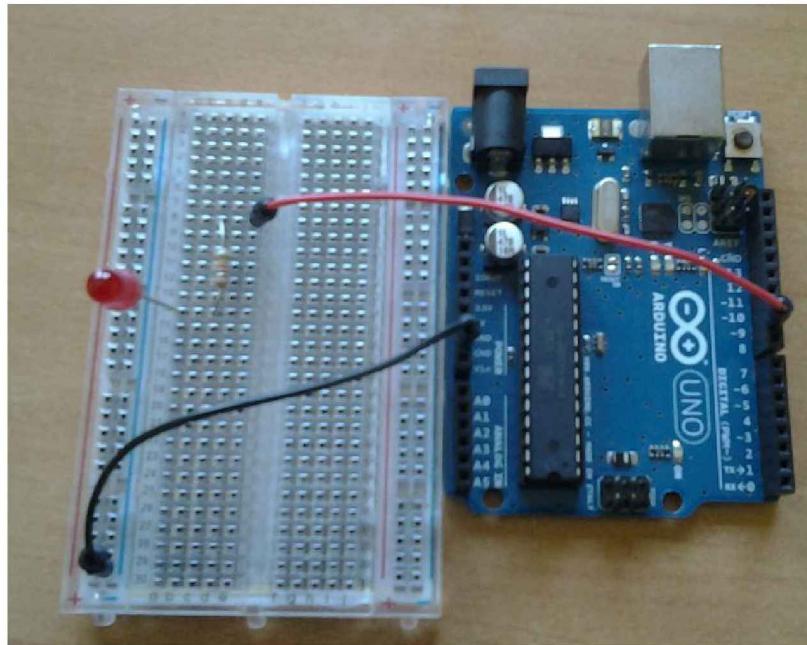


Figura 5.5: Proyecto LED desvaneciente final

5.4 // Botón

5.4.1 // Objetivo:

Encender un led a través de un pulsador.

La función `digitalRead()`, lee el valor de un pin digital especificado, los valores pueden ser HIGH (alto) o LOW (bajo). En virtud del valor proporcionado, sabremos si el botón está o no pulsado.

5.4.2 // Material necesario:

- Arduino.
- 1 placa de prototipado.
- 1 Led.
- 1 Resistencia 1k ohm.
- 1 Resistencia de 10k ohms.



Figura 5.6: Botón

- 1 Pulsador.
- Cables de conexionado.

5.4.3 // Conexionado:

Conecta un cable desde el pin digital 2 a una patilla del pulsador, esa misma patilla del pulsador se conecta a través de una resistencia pull-down (en este caso 10 KOhms) a masa. El otro extremo del pulsador se conecta a la fuente de 5 voltios.

Cuando el pulsador está abierto (sin pulsar) no hay conexión entre las dos patas del pulsador, de forma que el pin que está conectado a tierra (a través de la resistencia pull-down), leemos un LOW (bajo ó 0). Cuando el botón se cierra (pulsado), se establece la unión entre sus dos extremos, conectando el pin a 5 voltios, por lo que leemos un HIGH (alto ó 1).

Nota: Resistencias Pull UP y Pull Down: Se trata de una configuración de las resistencias. Esta configuración establece un estado lógico a la entrada de un circuito lógico cuando dicho circuito está en reposo, siendo Pull UP un estado alto y Pull Down bajo. Así se evitan falsos estados debidos al ruido eléctrico.

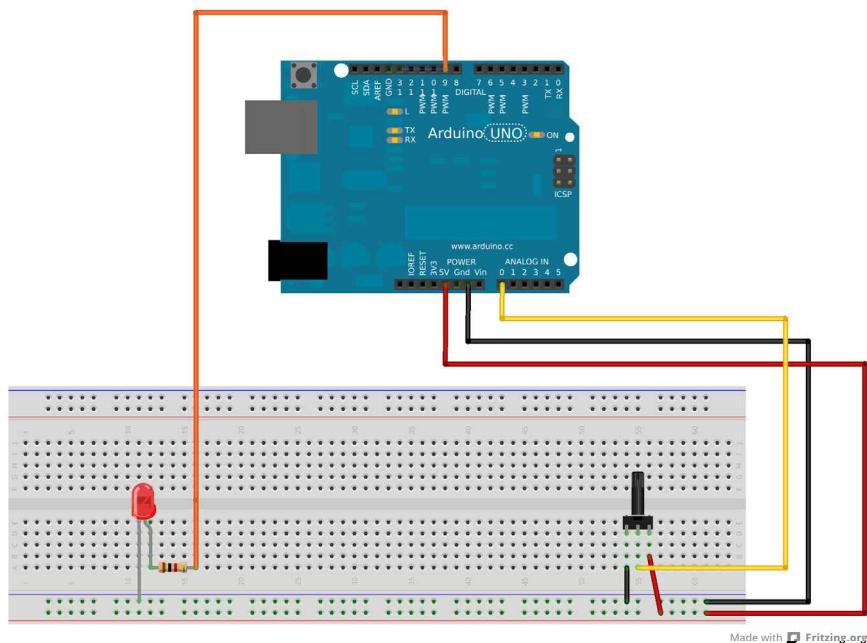


Figura 5.7: Prototipado Botón

5.4.4 // Sketch:

```
/*
  Ejemplo de uso de la función digitalRead, para
  controlar el valor de entrada de un pin digital.

*/
int buttonPin = 2;      // Pin donde conectamos el switch.
int ledPin = 13;        // Pin donde conectamos el led.

int buttonState = 0;    // variable para almacenar el estado del boton.

void setup() {
  pinMode(ledPin, OUTPUT); //Inicializamos el led como salida.
  pinMode(buttonPin, INPUT); //Inicializamos el boton como entrada.
}

void loop(){
  // leemos el estado del pin digital 2 y lo almacenamos
  // en la variable.

  buttonState = digitalRead(buttonPin);

  //Comprobamos si el boton esta pulsado.

  if (buttonState == HIGH) {
    // Si es verdadero, encendemos el led.

    digitalWrite(ledPin, HIGH);
  }
  else {
    // de lo contrario, lo apagamos.

    digitalWrite(ledPin, LOW);
  }
}
```

5.4.5 // Proyecto final:

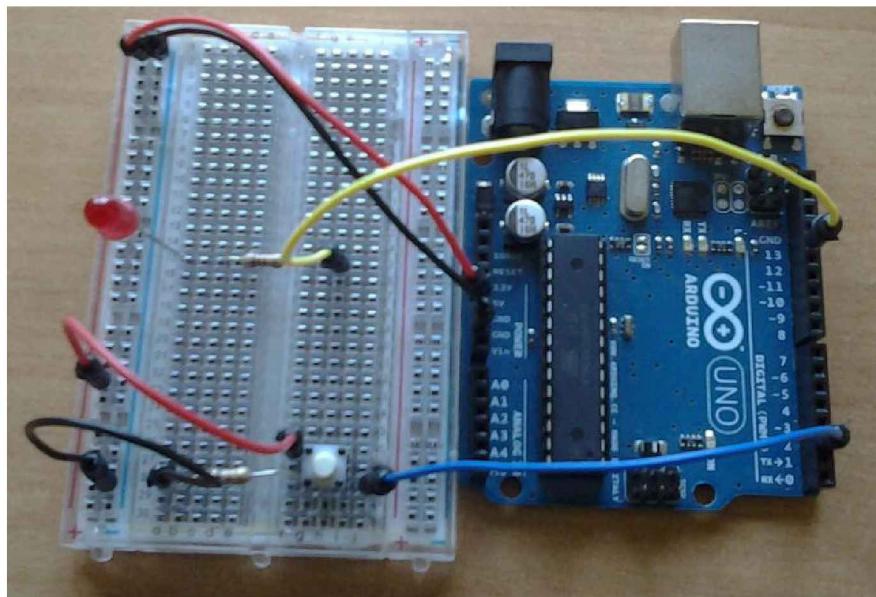


Figura 5.8: Proyecto Botón final

5.5 // Fadding por potenciómetro

5.5.1 // Objetivo:

Variar la luminosidad de un led en función de la intensidad leída en un potenciómetro.

La función `analogRead()`, lee el valor de tensión en el pin analógico especificado. La placa Arduino posee 6 canales conectados a un conversor analógico digital de 10 bits. Esto significa que convertirá tensiones entre 0 y 5 voltios a un número entero entre 0 y 1023.

Al girar el eje del potenciómetro cambiamos la resistencia, así al girar el eje totalmente hacia masa obtendremos 0 voltios (lectura 0), mientras que al girarlo hacia el lado de 5V obtendremos una lectura de 1023.

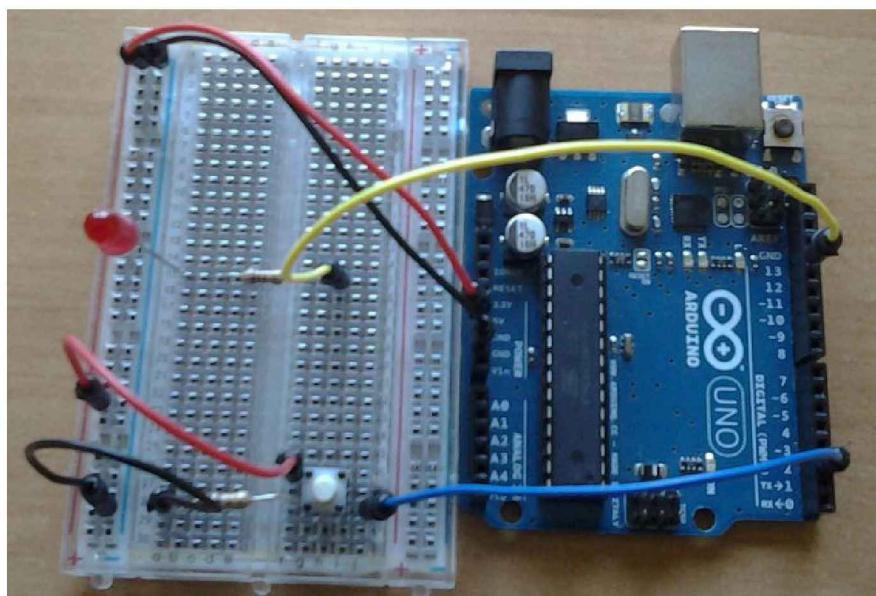


Figura 5.9: Potenciómetro

5.5.2 // Material necesario:

- Arduino.
- 1 Led.
- 1 Resistencia 1k ohm.
- 1 Potenciómetro de 1K ohm.
- Placa de prototipado.
- Cables de conexionado.

5.5.3 // Conexionado:

Conecta los dos extremos del potenciómetro a 5v y masa respectivamente, el elemento central del mismo conéctalo al pin A0 de Arduino.

Conecta el extremo corto del led a masa y el positivo a un extremo de la resistencia, el otro extremo de la misma la conectas al pin digital 9.

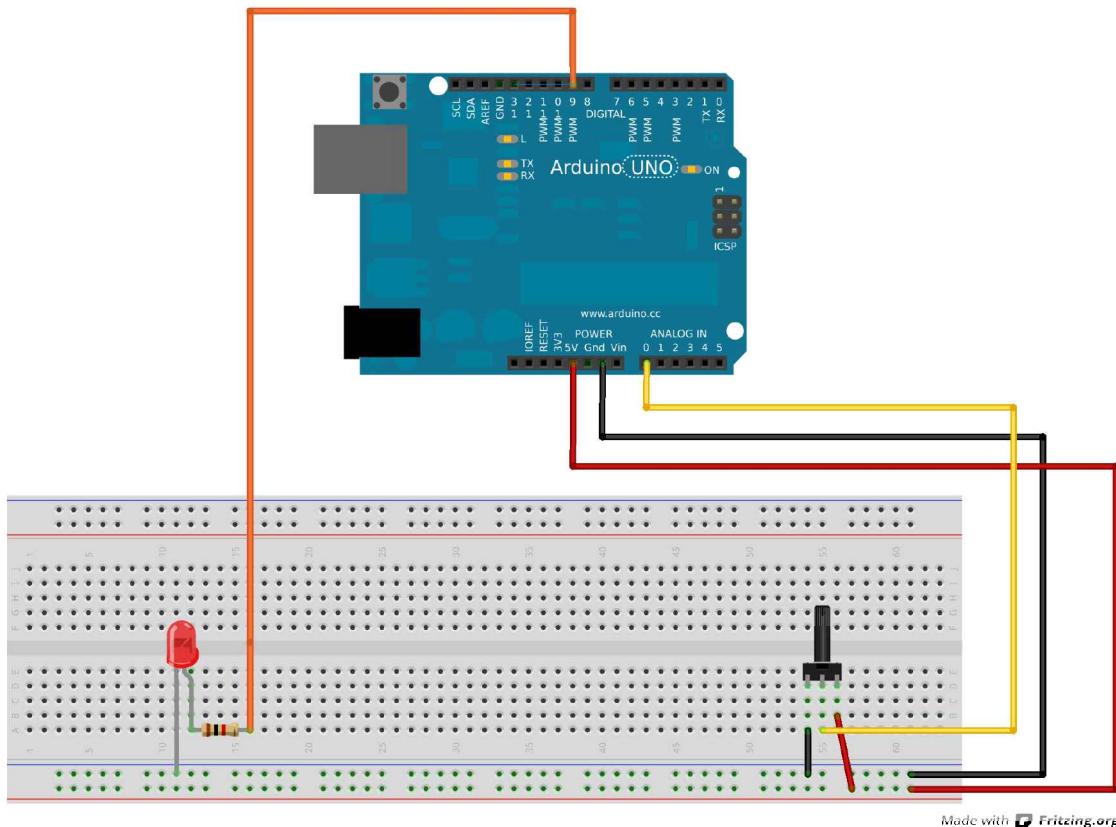


Figura 5.10: Prototipado Potenciómetro

5.5.4 // Sketch:

```
/*
Uso de la función analogRead() para leer el valor de tensión
en un pin analógico.

*/
int ledPin = 9;          // LED conectado al pin digital 9
int analogPin = 0;        // potenciómetro conectado al pin A0
int val = 0;              // variable en la que se almacena el dato leído

void loop(){
}

pinMode(ledPin, OUTPUT);    // establecemos ledPin como output
}

void loop()
{
    val = analogRead(analogPin);    // lee la tensión en el pin A0
    /*
    // los valores de analogRead van desde 0 a 1023 y los valores de
    // analogWrite van desde 0 a 255, por eso ajustamos el ciclo de trabajo
    // a el valor leído dividido por 4.

    */
    analogWrite(ledPin, val / 4);
}
```

5.5.5 // Proyecto final:

Aspecto del proyecto una vez terminado.

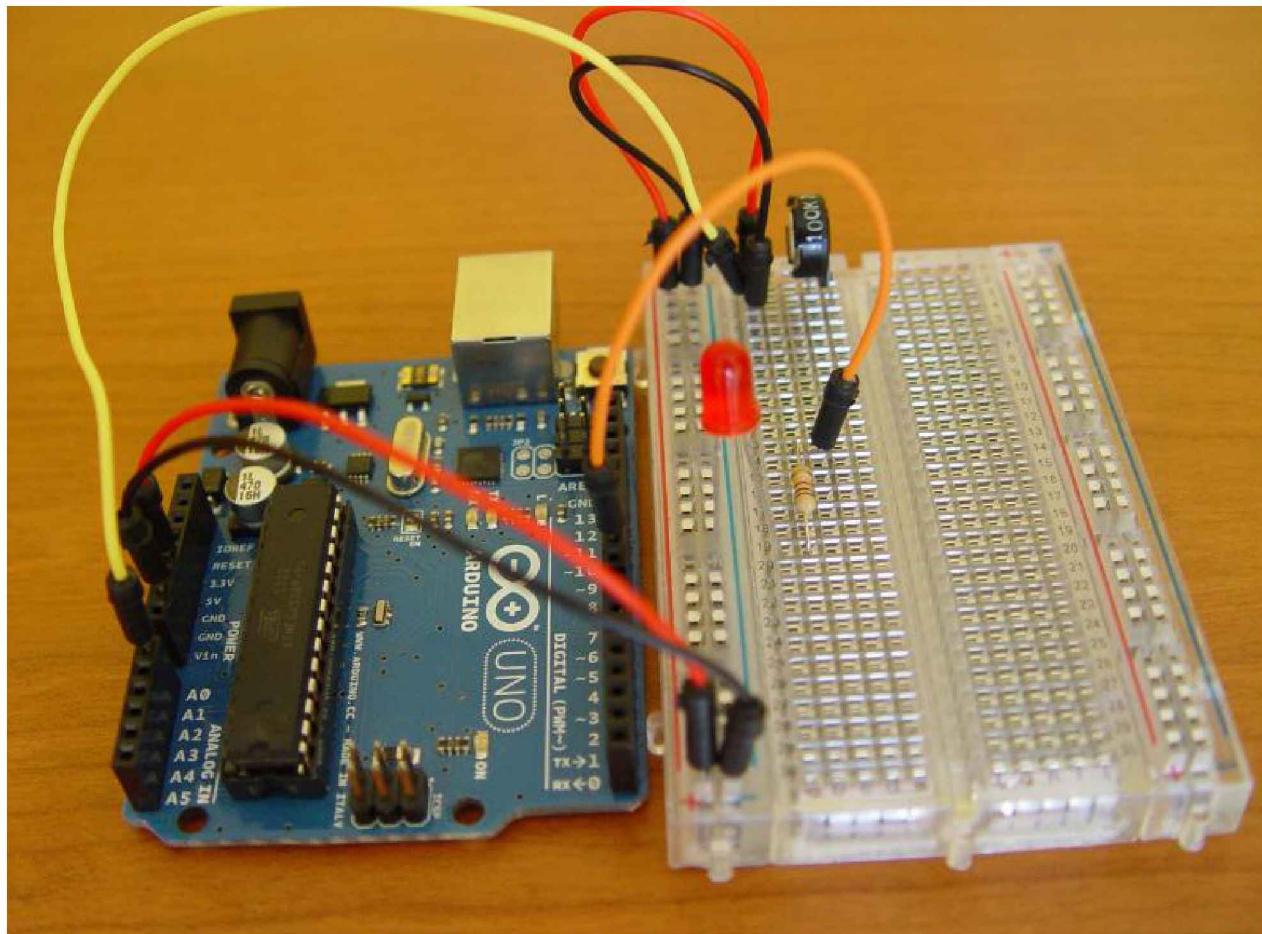


Figura 5.11: Proyecto final potenciómetro

Capítulo 6

Ejemplos

con Arduino

Avanzados I

6.1 // Tono

6.1.1 // Objetivo:

Reproducir una melodía simple en un zumbador.

La función `tone()`, genera una onda cuadrada de la frecuencia especificada en un pin. La duración puede ser especificada, en caso contrario la onda continua hasta que haya una llamada a `noTone()`. De esta manera, jugando con la frecuencia y la duración podremos crear nuestras propias melodías.

NOTA: las frecuencias audibles por el oído humano van de los 20Hz a los 20KHz por lo que el parámetro “frecuencia” debería estar comprendido entre estos dos valores.

6.1.2 // Material necesario:

- Arduino.
- 1 Zumbador (se puede conseguir en cualquier placa base de pc).
- Placa de prototipado.
- Cables de conexionado.



Figura 6.1: Buzzer

6.1.3 // Conexionado:

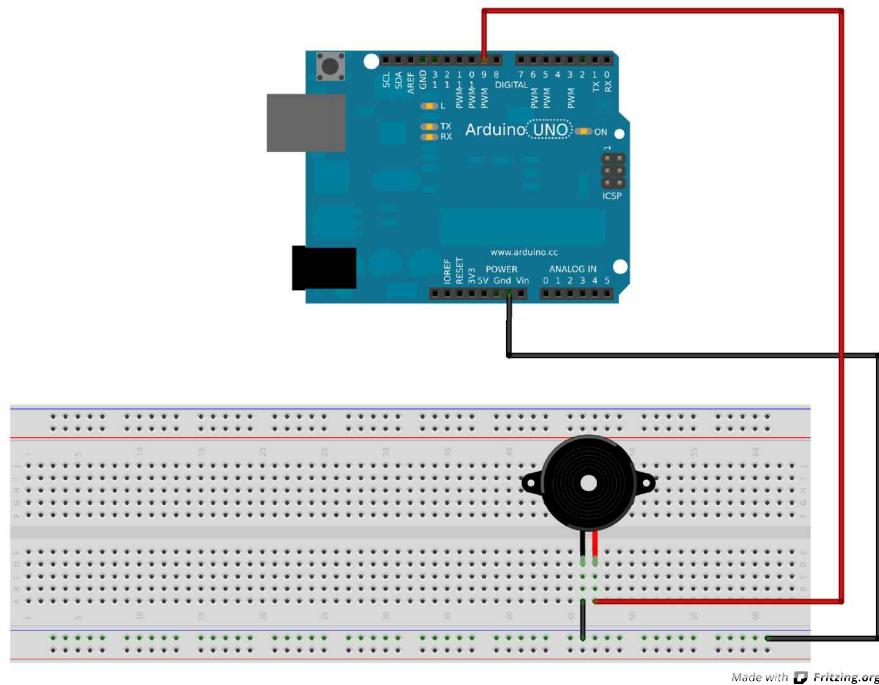


Figura 6.2: Conexiones buzzer

6.1.4 // Sketch:

```
//Simple sketch para hacer sonar un buzzer.
```

```
//El buzzer esta conectado al pin 9.
```

```
int analogPin = 9;
```

```
void setup() {  
    pinMode(buzzerPin, OUTPUT);  
}
```

```
void loop() {  
    // Reproduce una nota en el pin 9  
    // durante 150 ms.  
    tone(buzzerPin, 440);  
    delay(150);
```

Ejemplos con Arduino Avanzados I

```
// Detiene la funcion buzzer.  
noTone(buzzerPin);  
}  
  
// Reproduce otra nota durante 200 ms.  
tone(buzzerPin, 440);  
delay(200);  
}  
  
// Detiene la funcion  
noTone(buzzerPin);  
}  
  
// Reproduce otra nota durante 300 ms.  
tone(buzzerPin, 523);  
delay(200);  
  
}
```

6.1.5 // Proyecto final:

Aspecto final del conexionado.

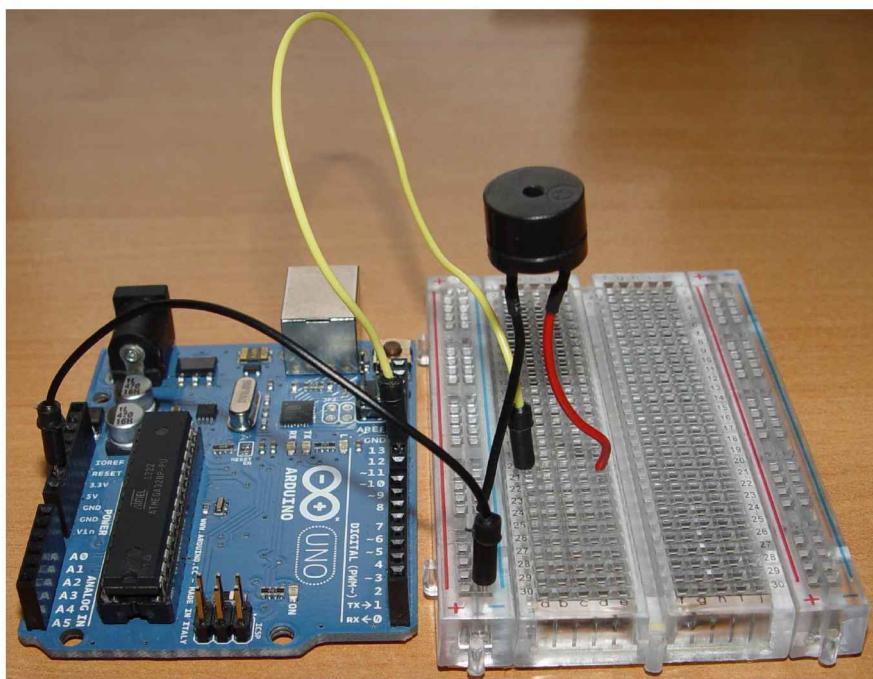


Figura 6.3: Proyecto final buzzer

6.2 // Sensor de luz

6.2.1 // Objetivo:

Encender o apagar un led en función de la intensidad de luz.

En este ejercicio conectaremos un sensor de luz (LDR). Un sensor LDR (Light Dependent Resistor), es una resistencia variable que varía su valor dependiendo de la cantidad de luz que incide sobre su superficie. A mayor intensidad, menor será su resistencia.

Usaremos también la comunicación serial para hacer las lecturas del valor del LDR.

La función map(), “mapea” un valor leído con la función analogRead (que suele variar entre 0-1023), a un valor que se necesita para los pines PWM que va de 0 a 255.

La función map() todavía puede devolver valores fuera de rango, por eso la función constrain(), “constríñe” los valores para que no se salgan del rango establecido.

Serial.begin(9600), inicializa el puerto serie a 9600 Baudios. Esto sirve para establecer la misma velocidad de comunicación entre dos dispositivos para el intercambio de datos.

Serial.print(): imprime en el puerto serie el valor de lo que se le pase como parámetro.
 Serial.println(), hace lo mismo pero con un salto de línea para que la información sea más clara.

6.2.2 // Material necesario:

- Arduino.
- 1 Led.
- 1 Fotorresistencia.
- 1 Resistencia 220 Ohm.
- 1 Resistencia de 10k Ohm.
- Placa de prototipado.
- Cables.

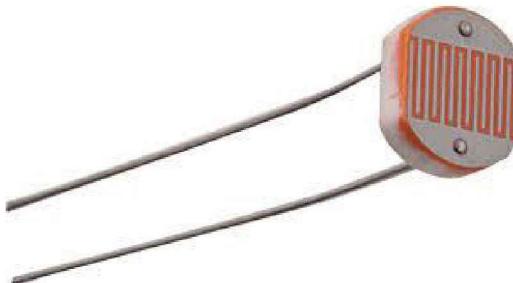


Figura 6.4: Sensor de luz

6.2.3 // Conexionado:

Conecta el polo negativo del led a masa y el positivo a un extremo de la resistencia de 220 Ohm, el otro extremo de la resistencia al pin 11 de Arduino. Conecta un extremo de la foto-célula a 5v., el otro extremo al pin analógico 0. Conecta la resistencia de 10k., entre el pin analógico 0 y masa (pull-down).

Nota: Arduino no puede medir resistencias sólo voltajes, por eso usamos un divisor de voltaje. El divisor de voltaje envía un voltaje alto cuando la fotocélula recibe gran luminosidad, y bajo cuando hay poca luz.

Un divisor de voltaje requiere que se conecte una fuente de voltaje a través de dos resistencias en serie, en este caso la resistencia de 10k + la propia fotocélula (no olvidemos que es una resistencia variable). Se trata de un circuito simple que reparte la tensión de una fuente entre una o más impedancias conectadas. Con sólo dos resistencias en serie y un voltaje de entrada, se puede obtener un voltaje de salida equivalente a una fracción del de entrada.

Nota: Es importante calibrar la fotoresistencia. Según el sketch que adjuntamos a continuación, leemos los valores “en bruto” y nos quedamos con el valor máximo entregado cuando en la estancia donde ubicamos la fotocélula está al máximo de iluminación, y el valor mínimo que leemos cuando tapamos la fotocélula.

Estos valores serán las variables min y max.

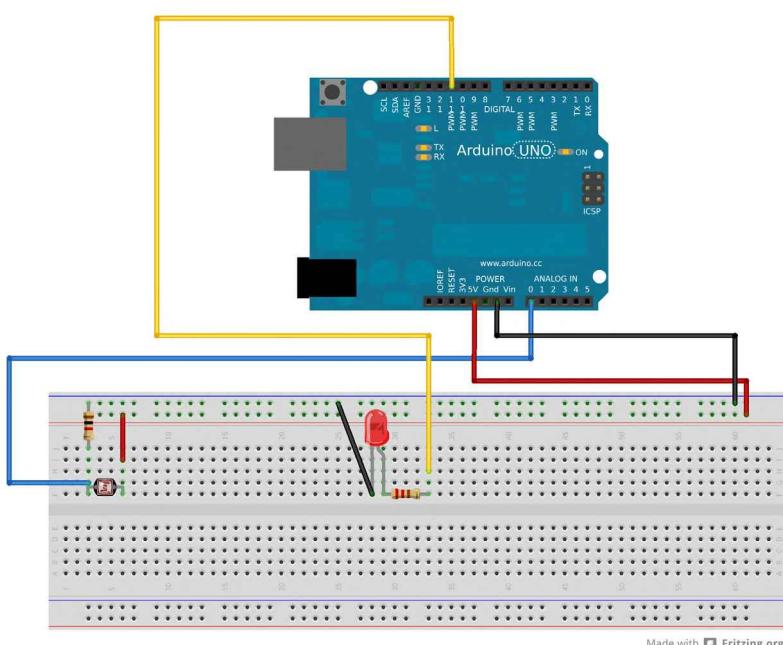


Figura 6.5: Sketch Sensor de Luz

6.2.4 // Sketch:

```
/*
Hacemos brillar un led según la intensidad
recibida por la fotocelula.

*/
int photoSensor = 0; // Pin analogico para la fotoresistencia
int ledPin=11; // Pin al que se conecta el LED
```

```
int valor; //define una variable en la que haremos los cálculos
int min = 780; //Variable de calibrado valor mínimo.
int max = 940; //Variable de calibrado valor máximo.

void loop(){
}

pinMode( ledPin, OUTPUT ); // Establece el pin del LED como salida.
Serial.begin(9600); // Establecemos la comunicación serie a 9600 Baudios.

}

void loop()
{
valor = analogRead(photoSensor); //Leemos el valor de la fotocelula.
valor = map(valor, min, max, 0, 255); //Mapeamos el valor al rango 0-255.
valor = constrain(valor, 0, 255); //Evitamos valores fuera de rango.
analogWrite(ledPin, valor); //Enviamos el valor analógico al pin 11.

Serial.println(valor); //Enviamos el valor al puerto serie.

// Hacemos una pequeña pausa.

delay(100);
}
```

Para ver el valor enviado al puerto serie, pincha en el ícono “Monitor serie”:



Figura 6.6: Monitor serial

6.2.5 // Proyecto final:

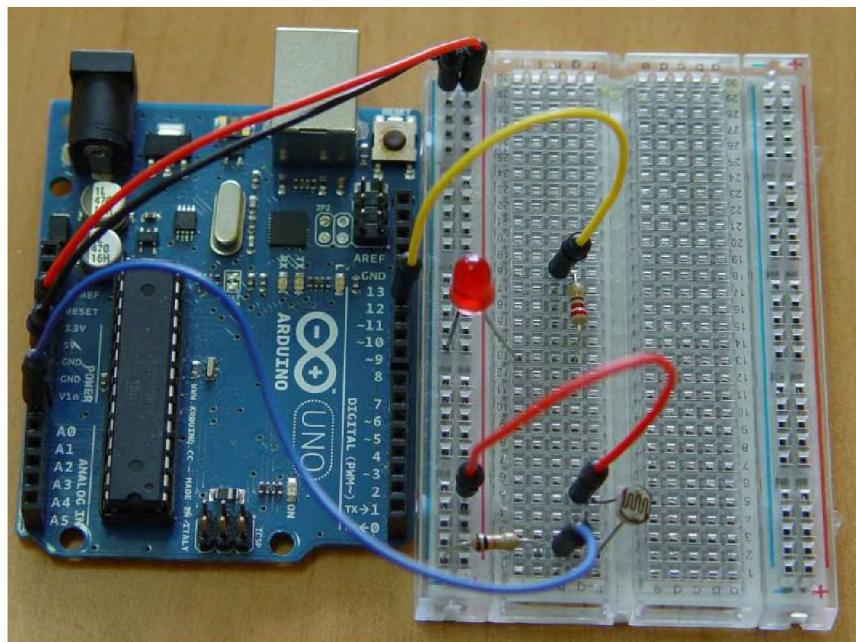


Figura 6.7: Resultado final serial para el sensor de luz

6.3 // Sensor de temperatura

6.3.1 // Objetivo:

Leer la temperatura ambiente y mostrarla por el puerto serie. Para esto usaremos el sensor LM35. La tensión de salida de este sensor es directamente proporcional a la temperatura en grados centígrados (cada grado centígrado equivale a 10 mV). No requiere de ninguna calibración externa, y su rango de funcionamiento está entre -55 y 150°.

El sensor LM35 tiene 3 pines: positivo (5v.), negativo y señal.

6.3.2 // Material necesario:

- Arduino.
- Sensor LM35
- Placa prototipos.
- Cables.

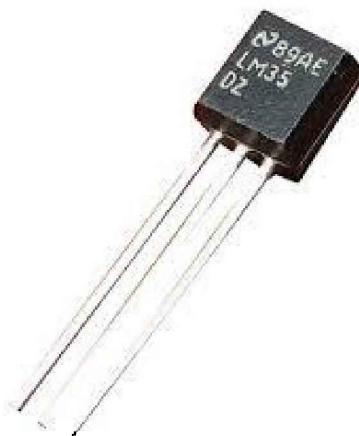


Figura 6.8: Sensor de temperatura

6.3.3 // Conexionado:

Conecta un cable desde el pin de 5v de Arduino al pin positivo del sensor, otro desde el pin negativo del sensor al GND de Arduino, y el tercero desde el pin A0 de Arduino al pin de señal del sensor.

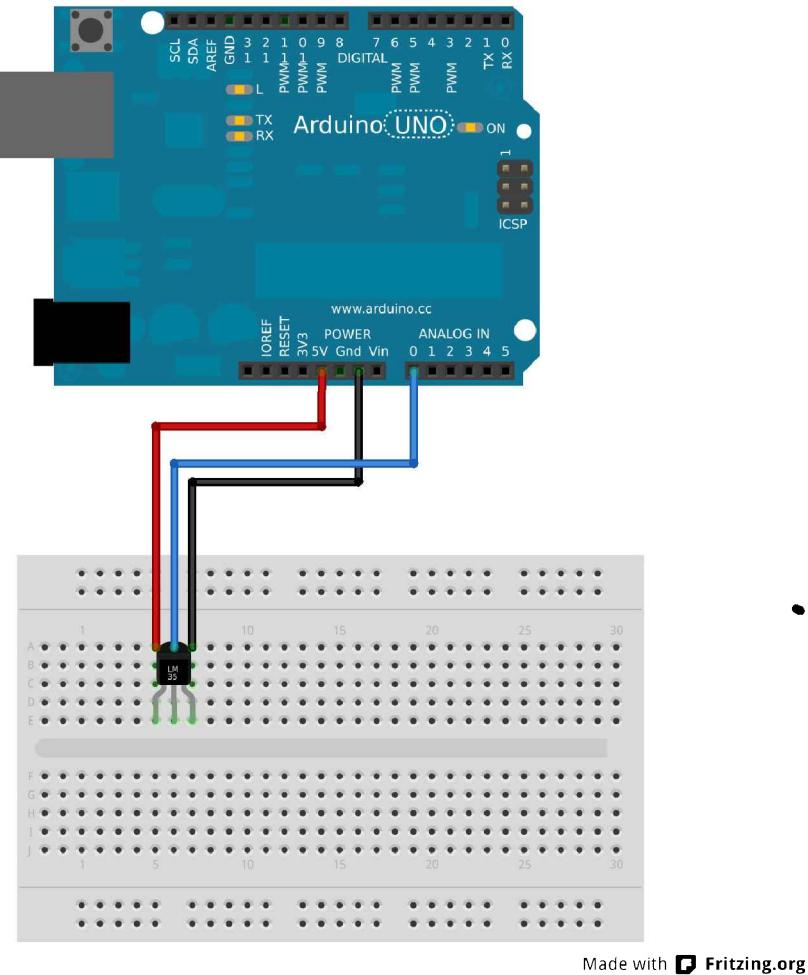


Figura 6.9: Sketch Sensor de Temperatura

6.3.4 // Sketch:

```
/*
Sketch para medir la temperatura
con el sensor de temperatura LM35
e imprimirla por el puerto serie.
*/
```

```
float temp; //Variable donde almacenaremos el valor medido.
int tempPin = 0; // Definimos la entrada en pin A0
```

Ejemplos con Arduino Avanzados I

```
void setup()
{
    // Abre puerto serial y lo configura a 9600 bps
    Serial.begin(9600);
}

void loop()
{
    // Leemos el valor del sensor.
    temp = analogRead(tempPin);

    // Calcula la temperatura usando como referencia 5v.
    temp = (5.0 * temp * 100.0)/1024.0;

    // Envia el dato al puerto serie.
    Serial.print(temp);
    Serial.print(" grados Celsius\n");

    // Una pequeña pausa hasta la siguiente lectura.
    delay(3000);
}
```

Una vez subido el sketch, pulse en el icono del “Monitor serie” para visualizar las lecturas que nos entrega el sensor.

6.3.5 // Proyecto final:

Aspecto del proyecto ya montado.

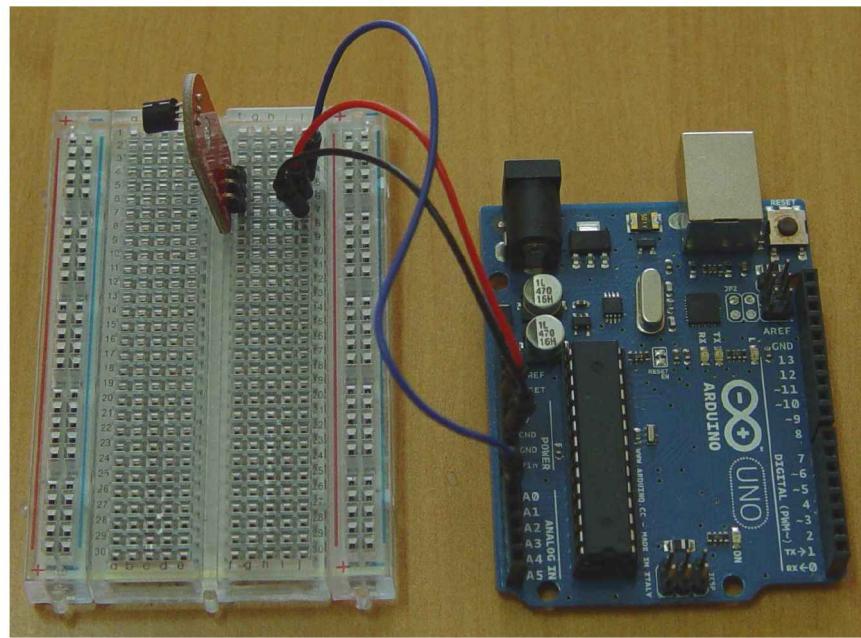


Figura 6.10: Proyecto Final Sensor de Temperatura

6.4 // Sensor de temperatura y humedad

6.4.1 // Objetivo:

En este ejercicio vamos a ver como hacer funcionar el sensor DHT11 que nos va a servir para medir tanto la humedad relativa como la temperatura. Este sensor puede medir la humedad entre el rango 20% – aprox. 95% y la temperatura entre el rango 0°C – 50°C. Lo malo de este sensor es que solo nos va a dar medidas enteras, es decir sin decimales, ya que la resolución que presenta es de 1% para la humedad relativa y de 1°C para la temperatura. La buena noticia es que es un sensor muy económico y que nos permite obtener medidas tanto para humedad como para temperatura.

Explicaremos además como agregar una nueva librería al IDE de Arduino.

6.4.2 // Material necesario:

- Arduino.
- Sensor DHT11
- Placa prototipos.
- Cables.

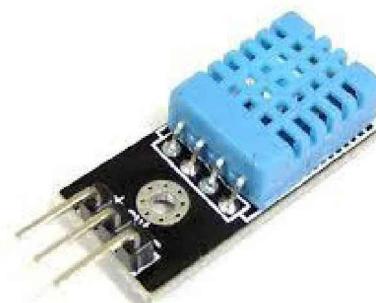


Figura 6.11: Sensor de Temperatura y Humedad

6.4.3 // Conexionado:

Conectamos +5v y el negativo en los pines izquierdo y derecho del sensor respectivamente (comprobar las coincidencias en otros fabricantes). El pin central lo conectamos a la placa de Arduino en el pin digital 2.

¿Porqué en un pin digital cuando hasta ahora los sensores se conectaban en los analógicos?

Esto se debe a que el sensor DHT11 emite datos, no solo estados como los anteriores. Se trata de un sensor inteligente el cual integra un microcontrolador, éste mide la humedad y temperatura y la transforma en un valor digital que entrega por el pin central.

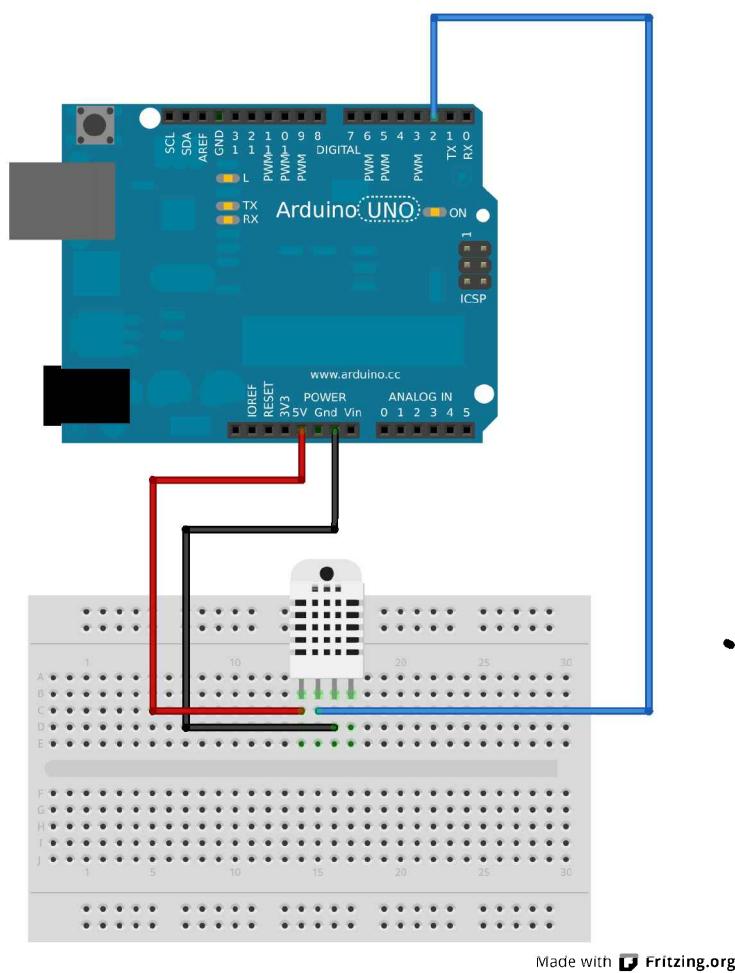


Figura 6.1: Sketch Sensor de Temperatura y Humedad

6.4.4 // Sketch:

Antes de ponernos manos a la obra con el sketch, vamos a explicar como agregar una nueva librería al IDE de Arduino.

La función de las librerías es proveer funcionalidad extra a nuestro sketch. Por defecto el IDE de Arduino ya viene con varias librerías preinstaladas, pero si necesitamos instalar una

nueva, simplemente hay que descargarla y descomprimirla en una carpeta propia dentro de la carpeta “libraries”, que se encuentra dentro de la carpeta donde está instalado el IDE de Arduino.

Para este ejercicio, copie la carpeta “DHT” que está dentro de la carpeta “Sketches”, dentro de la carpeta “libraries” de la instalación de Arduino. Antes de seguir, reinicie el IDE de Arduino.

```
/*
Ejemplo de sketch para los sensores DHT.
Written by ladyada, public domain.

*/
#include "DHT.h" //Incluimos la libreria que acabamos de instalar.

#define DHTPIN 2      // Definimos el pin digital 2 como de entrada.
#define DHTTYPE DHT11 // Establecemos el sensor de tipo DHT 11.
DHT dht(DHTPIN, DHTTYPE); //Creamos el objeto dht.

void setup() {
    Serial.begin(9600);
    //Inicializamos la libreria.
    dht.begin();
}

void loop() {
    //Llamamos a los metodos de lectura y almacenamos el valor en su
    //correspondiente variable.
    int h = dht.readHumidity();
    int t = dht.readTemperature();

    // Si los resultados son NaN (not a number), algo va mal.
    if (isnan(t) || isnan(h)) {
        Serial.println("Fallo de lectura del DHT");
    }
}
```

Ejemplos con Arduino Avanzados I

```
 } else {  
 //De lo contrario, imprimimos el valor por el puerto serie.  
 Serial.print("Humedad: ");  
 Serial.print(h);  
 Serial.print(" %\t");  
 Serial.print("Temperatura: ");  
 Serial.print(h);  
 Serial.println(" °C");  
 delay(3000); //Hacemos una pausa de 3 segs. entre cada lectura.  
 //Se trata de un sensor lento. No bajar de este valor.  
 }  
 }
```

Una vez subido el sketch, pulsa en el icono del “Monitor serie” para visualizar las lecturas que nos entrega el sensor.

6.4.5 // Proyecto final:

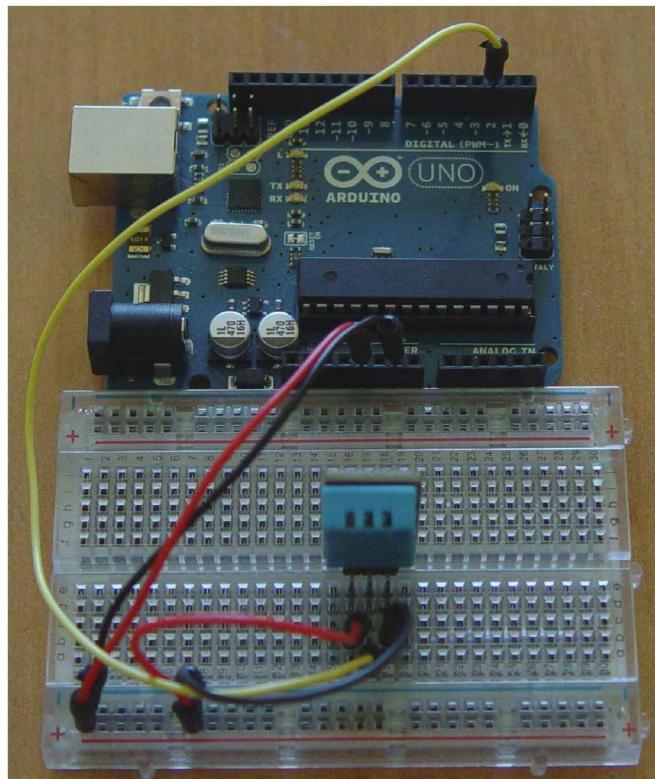


Figura 6.10: Proyecto Final Sensor de Temperatura

Capítulo 7

Ejemplos

con Arduino

Avanzados II

7.1 // Servo

7.1.1 // Objetivo:

En este ejercicio conectaremos un servo a un potenciómetro, así al cambiar el valor del potenciómetro, moveremos el servo a la posición indicada por el mismo.

Un servo, a diferencia de un motor convencional, puede moverse a una posición exacta variando el pulso del voltaje (el servo gira en un ángulo de 0 a 180°). Ya hemos visto en ejercicios anteriores como usar los pines PWM (pulse width modulation), en este ejercicio volveremos a usarlos pero ahora para controlar la rotación de un servo. Por ejemplo: un pulso de 1.5 milisegundos, hace girar el servo 90°.

También repasaremos cómo añadir librerías al IDE de Arduino, en este caso la librería “servo” que añade nuevas funcionalidades a nuestro IDE.

7.1.2 // Material necesario:

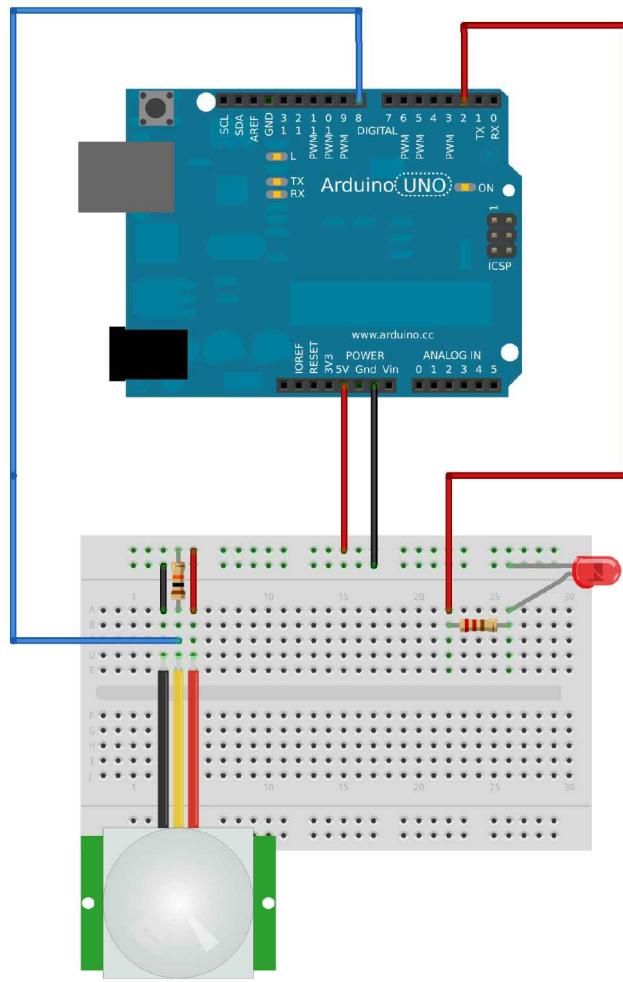
- Arduino.
- Servo (el económico SG90 nos vale perfectamente para este ejercicio).
- Potenciómetro de 100K.
- Placa prototipos.
- Cables.



Figura 7.1: Sensor de movimiento

7.1.3 // Conexionado:

Conecta +5v. y el negativo a ambos extremos del potenciómetro, conecta el conector central del mismo al pin analógico 0 de Arduino. Conecta +5v. al cable rojo del servo y el negativo al cable castaño. El cable amarillo del servo (control), conéctalo al pin 9 de la placa Arduino.



Made with Fritzing.org

Figura 7.2: Sketch Sensor de Movimiento

7.1.4 // Sketch:

```
/*
Sketch que mueve un servo según la posicion
de un potenciómetro.

*/
#include <Servo.h> //Incluimos la libreria servo.

Servo myservo; //Creamos el objeto servo.
```

```
int potPin = 0; // potenciómetro conectado al pin A0
```

Ejemplos con Arduino Avanzados II

```
int valPot = 0; //Valor del potenciómetro.  
  
void setup()  
{  
    myservo.attach(9); //Configuramos el servo en el pin 9.  
}  
  
void loop()  
{  
    valPot = analogRead(potPin); //Leemos el valor del potenciómetro.  
    valPot = map(valPot, 0, 1023, 0, 180); //Mapeamos el valor al rango 0-180.  
    valPot = constrain(valPot, 0, 180); //Evitamos valores fuera de rango.  
  
    myservo.write(valPot);  
    //Movemos el servo a la posición que marca el potenciómetro.  
}
```

7.1.5 // Proyecto final:

Aspecto del proyecto una vez terminado.

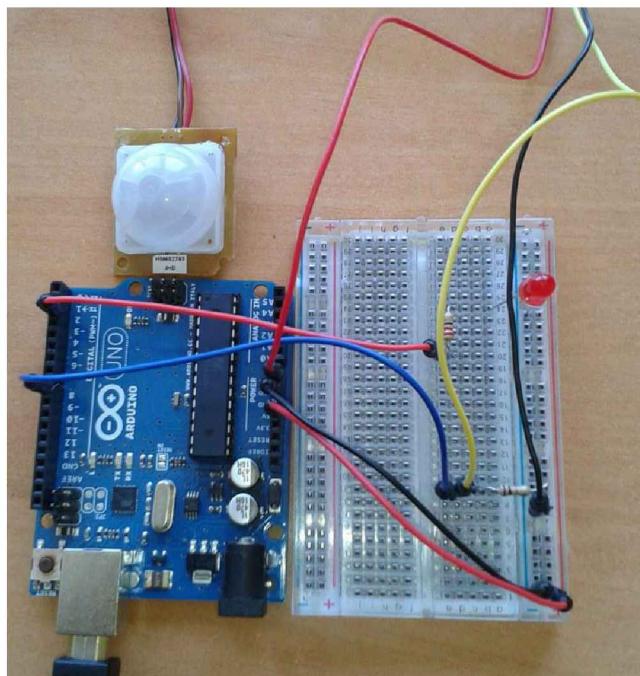


Figura 7.3: Proyecto Final Sensor de Movimiento

7.2 // Sensor de movimiento

7.2.1 // Objetivo:

En este ejercicio vamos a conectar un sensor de movimiento PIR a nuestra placa Arduinvo. Cuando detecte movimiento encenderemos un led.

Este dispositivo nos será útil para activar cámaras, alarmas, etc., al detectar presencia.

Su funcionamiento consiste en escanear un área para detectar movimiento en ella. Si se produce, pone el pin de control en un nivel bajo.

7.2.2 // Material necesario:

- Arduino.
- Placa prototipos.
- Sensor de movimiento (SE-10).
- Led.
- Resistencias de 220 Ohm y 10Kohm.
- Cables de conexión.

7.2.3 // Conexionado:

El sensor posee 3 cables: positivo (de 5 a 12v.), negativo y pin de alarma en colector abierto, esto significa que, cuando detecte movimiento, pasará a nivel bajo. Además necesitamos conectar una resistencia pull-up entre +5v. y el pin de alarma.

Nota: Si se detectase un comportamiento errático en el sensor, es conveniente suministrarle algo más de tensión (parece que funciona mejor a partir de 7v.). para esto conectamos la placa Arduino a un alimentador externo (sobre 9v.), y el cable positivo del sensor al pin “Vin” de Arduino, así conseguimos alimentar el sensor también a 9 voltios.

7.2.4 // Sketch:

```
/*
Sketch que enciende un led cuando se detecta
movimiento.

*/
```

Ejemplos con Arduino Avanzados II

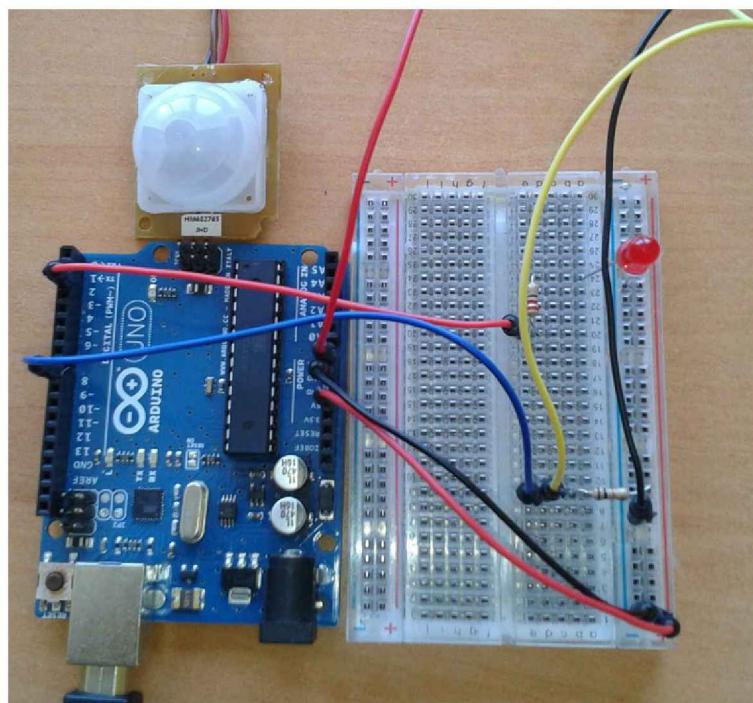
```
int motionPin = 8; //Pin de conexion del sensor.  
int lightPin = 2; //Pin de conexion del led.
```

```
void setup(){  
    pinMode (motionPin, INPUT);  
    pinMode (lightPin, OUTPUT);  
    delay(3000); //Esperamos 3 segs. para que el sensor se calibre.  
}
```

```
void setup(){  
    digitalWrite(lightPin, LOW);  
    delay(1000);  
    int sensorValue = digitalRead(motionPin); //Hacemos una lectura del sensor.
```

```
    if (sensorValue == LOW) //Movimiento detectado.  
    {  
        digitalWrite(lightPin,HIGH); //Encendemos led.  
        delay(3000); //Mantenemos el led encendido 1 segundo.  
    }  
}
```

7.2.5 // Proyecto final:



7.3 // Mini estación meteorológica

7.3.1 // Objetivo:

Poner en práctica los conceptos aprendidos en capítulos anteriores, para ello vamos a desarrollar una mini estación meteorológica. Básicamente lo que hará será mostrarnos en una pantalla de cristal líquido, la humedad relativa y temperatura. Dejamos a la elección del alumno la posibilidad de conectarle a mayores los sensores que estime oportuno. Tomaremos este ejercicio como un punto de partida.

En este proyecto vamos a usar un display LCD 20x4 que conectaremos mediante el protocolo I2C. Para ello usaremos una placa para la conexión del lcd por I2C, las cuales son bastante económicas, muy fáciles de conectar (los pines de la placa van a los pines correspondientes del lcd), y además incluyen un potenciómetro para regular el contraste.

El protocolo I2C es un bus de comunicaciones en serie que utiliza dos líneas para transmitir la información: una para datos y otra para la señal de reloj. En el Arduino Uno, la línea de datos (SDA) está en el pin analógico 4, y la línea de reloj (SCL), en el pin analógico 5. Con este protocolo podemos ahorrar muchos pines que podríamos necesitar para conectar más sensores a nuestro proyecto.

La librería que usaremos para la comunicación sobre este protocolo es la Wire.h, que Arduino ya tiene de serie.

Además usaremos una nueva librería “LiquidCrystal_I2C.h” que nos permitirá controlar el display LCD por el protocolo I2C de una forma muy sencilla. Una vez incluida la librería, con “lcd.begin()” inicializamos el LCD pasándole como parámetros, el número de filas y columnas, “lcd.setCursor()” sitúa el cursor en la posición del LCD que se le pasará como parámetro, y por último “lcd.print()” imprimirá un texto o el valor de una variable que se le pasa como parámetro.

Para finalizar, incluimos la librería DHT.h, de la cual ya hablamos en un capítulo anterior.

7.3.2 // Material necesario:

- Arduino Uno.
- Placa de prototipado.
- Sensor LM35.
- Sensor DHT11.
- Display LCD 16x2 ó 20x4.
- Módulo I2C para LCD.

7.3.3 // Conexionado:

A parte del LCD y su módulo I2C, usaremos dos sensores: el LM35 y el DHT11, la razón de usar los dos es que el primero nos ofrece un rango de valores térmicos más amplios que el segundo, además de una precisión mayor.

Manos a la obra: Conectamos el módulo I2C al display lcd, y los cuatro pines de éste, de la siguiente manera:

- GND: a masa.
- VCC: 5 voltios.
- SDA: al pin A4 de Arduino.
- SCL: al pin A5 de Arduino.

El resto de conexiones, según se observa en el diagrama adjunto:

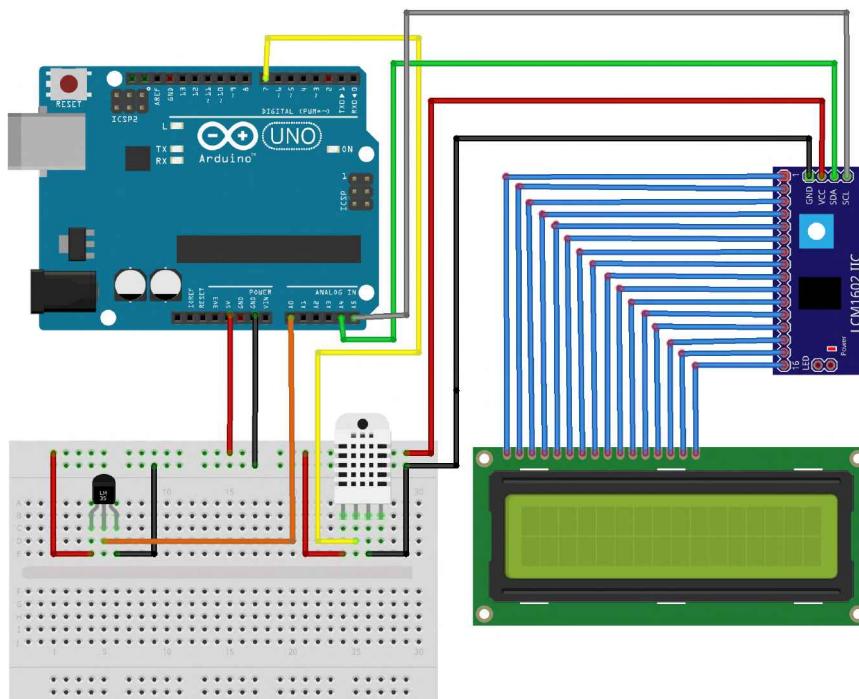


Figura 7.4: Diagrama Estación Meteorológica

Vamos a explicar brevemente los pines del lcd:

- Pin de selección de registro (RS) que controla en qué parte de la memoria del LCD está escribiendo datos. Puede seleccionar bien el registro de datos, que mantiene lo que sale en la pantalla, o un registro de instrucción, que es donde el controlador del LCD busca las

instrucciones para saber cuál es lo siguiente que hay que hacer.

- Pin de lectura/escritura (R/W) que selecciona el modo de lectura o el de escritura.
- Pin para habilitar (enable) que habilita los registros.
- 8 pines de datos (D00-D07). Los estados de estos pines (nivel alto o bajo) son los bits que está escribiendo a un registro cuando escribe, o los valores de lectura cuando está leyendo.
- Pin de contraste del display (Vo), pines de alimentación (+5V y GND) y pines de retro-iluminación (Bklt+ y Bklt-), que le permiten alimentar el LCD, controlar el contraste del display, o encender y apagar la retro-iluminación, respectivamente.

7.3.4 // Sketch:

```
#include <Wire.h> //Libreria para comunicación con el protocolo I2C.
#include <LiquidCrystal_I2C.h> //Libreria para el manejo del LCD.
#include "DHT.h" //Libreria para el sensor de humedad.
```

```
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
//Pines del lcd (ver diagrama adjunto).
// Addr, Enable, Rw, Rs, d4, d5, d6, d7, backlighpin, polarity
```

```
#define DHTPIN 7
#define DHTTYPE DHT11
float temp;
int tempPin = 0;
```

```
DHT dht(DHTPIN, DHTTYPE);
```

```
void setup() {
    lcd.begin(20, 4); //Inicializamos el LCD (20 filas x 4 columnas).
    dht.begin(); //Inicializamos el sensor de humedad.
}
```

Ejemplos con Arduino Avanzados II

```
void loop() {  
    int h = dht.readHumidity(); //Leemos la humedad relativa.  
    temp = analogRead(tempPin); //Leemos la temperatura.  
    temp = (5.0 * temp * 100.0)/1024.0;  
  
    if (isnan(h)) {  
        lcd.setCursor(0, 0); //Situamos el cursor en la fila 0 y columna 0.  
        lcd.print("Fallo de lectura"); //Imprimimos el mensaje en el LCD.  
        lcd.setCursor(0, 1); //Acabamos el mensaje en la siguiente fila.  
        lcd.print("del DHT");  
    }  
    else {  
        lcd.setCursor(0, 0);  
        lcd.print("Humedad: ");  
        lcd.setCursor(0, 1);  
        lcd.print(h);  
        lcd.setCursor(0, 1);  
        lcd.print("%");  
        lcd.setCursor(0, 0);  
        lcd.print("Temp: ");  
        lcd.setCursor(0, 1);  
        lcd.print(temp);  
        lcd.setCursor(0, 1);  
        lcd.print("del DHT"); //Imprime el simbolo de grados.  
        delay(3000);  
    }  
}
```


Capítulo 8

Node JS y

el Puerto Serial

8.1 // Introducción

La definición de Node JS según su página oficial es:

“Node.js es una plataforma basada en la JavaScript en tiempo de ejecución para que sea fácil la creación de aplicaciones de red rápidas y escalables. Node.js utiliza un modelo de no bloqueo orientado a eventos, I / O que lo hace ligero y eficiente, ideal para aplicaciones en tiempo real de datos intensivos que se ejecutan a través de dispositivos distribuidos.”

Para programar en Node JS es necesario conocer Javascript.

Las pruebas que se realizan en este curso se hacen desde web, por ello primero se estudia un framework muy popular llamado Express JS.

8.2 // Express JS

Para instalarlo se usa npm:

```
sudo npm install -g express-generator
```

La “g” significa que se instala de forma global con lo que se podrá llamar desde cualquier directorio. Al escribir:

```
express -V
```

Aparece el número de versión.

Por defecto como motor de templates usa Jade (<http://jade-lang.com/>), este lenguaje está muy bien, pero no es objetivo de este curso aprenderlo, por ello se usará EJS (<http://embeddedjs.com/>) que es mucho más sencillo ya que la base es HTML.

Primera aplicación con Express:

```
express -e myapp  
cd myapp && npm install
```

Desde dentro del directorio se ejecuta:

```
bin/www
```

Al entrar en <http://localhost:3000> se podrá ver la página web en funcionamiento.

Si se cambia el código de la aplicación (no el de los templates) es necesario parar el demonio y volver a arrancarlo. Para evitar hacer esto lo mejor es usar nodemon.

```
sudo npm install -g nodemon
```

Al terminar al escribir nodemon -V se verá la versión.

Esta es la estructura básica de la carpeta de la aplicación.

```
— app.js  
— bin  
  └── www  
— node_modules ...  
— package.json  
— public  
  └── images  
    ├── javascripts  
    ├── stylesheets  
    └── style.css  
— routes  
  └── index.js  
  └── users.js  
— views  
  └── error.ejs  
  └── index.ejs
```

Node JS y el Puerto Serial

- app.js es el archivo principal
 - bin es la carpeta de ejecutables
 - node_modules es donde se almacenan todos los paquetes que necesita la aplicación
 - package.json sirve para llevar un control de las dependencias y añadir scripts
 - public es una carpeta accesible desde la red, por ello se guardan las imágenes, los css y los javascripts.
-
- routes aquí estarían las estructuras de control
 - views los templates en los que se basa la web

Por comodidad modificamos la línea scripts dentro de package.json para empezar el programa.

```
"start": "nodemon -e js,css,ejs ./bin/www"
```

La letra “-e” significa que cuando se modifique cualquier archivo dentro del directorio con esa lista de extensiones nodemon se reiniciará.

```
npm start
```

Para modificar los archivos se usa gedit. Se instala desde el terminal con:

```
apt-get install gedit
```

Si necesitamos un nuevo módulo, como por ejemplo “reload”, lo añadimos a package.json. Dentro de dependencias:

```
"reload": ""
```

Se instala ejecutando:

```
npm install
```

Si se entra en el directorio node_modules se ve una carpeta llamada reload.

Para activar reload se necesitan copiar las librerías a un lugar accesible vía web:

```
cp node_modules/reload/lib/sockjs-0.3-min.js public/javascripts/
```

```
cp node_modules/reload/lib/reload-client.js public/javascripts/
```

Añadir los scripts en el archivo index.ejs

```
<script src="javascripts/sockjs-0.3-min.js"></script>  
<script src="javascripts/reload.js"></script>
```

Añadir la librería en app.js

```
var reload = require('reload');
```

Y por último en bin/www añadir la función reload.

```
reload(server, app);
```

Con esto se obtiene un entorno de trabajo cómodo ya que al guardar un archivo se recarga el navegador.

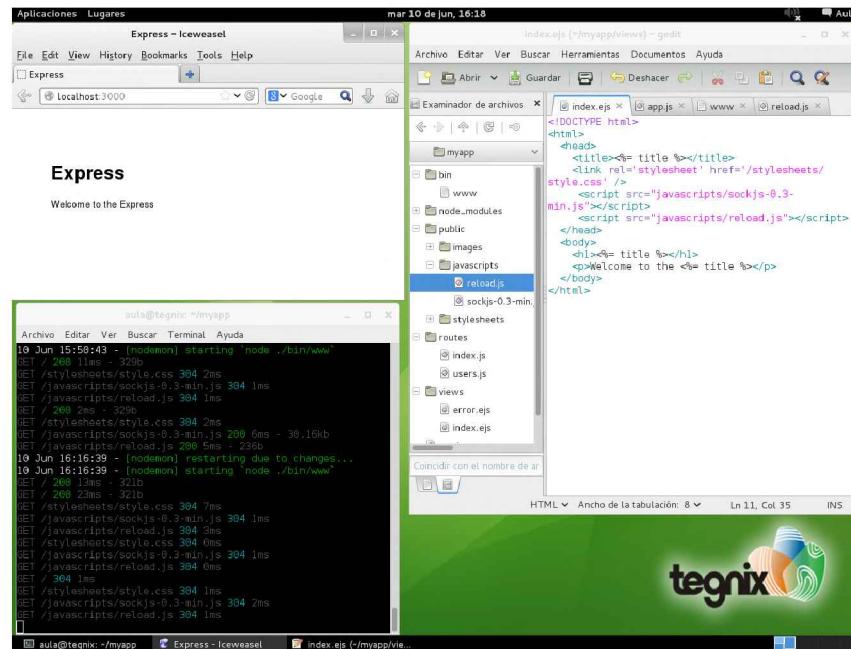


Figura 8.1: Captura pantalla Express JS

8.3 // Sockets

En el ejemplo anterior ya se ha usado la comunicación por socket, de hecho, al comprobar que la comunicación por socket se perdía, automáticamente se reiniciaba el navegador.

Ahora se profundiza en el estudio de socket y para ello se empieza con una nueva aplicación:

```
express -e sockets  
cd sockets
```

Se usan dos librerías: socket.io y socket.io-client. Entonces se añade al archivo package.json:

```
"socket.io": "",  
"socket.io-client": ""
```

Y se instala:

```
npm install
```

Se copia la librería para el cliente web:

```
cp node_modules/socket.io-client/socket.io.js public/javascripts/
```

Se añade al archivo index.js:

```
<script src="javascripts/socket.io.js"></script>
```

Analizando el archivo dentro de route index.js se puede ver como la función tiene dos parámetros req y res. En la variable res se guardan datos de todo tipo. Si añadimos al archivo:

```
console.log(res);
```

En el terminal se verá un listado en JSON con todos los valores de esta variable. Como queremos saber el "host" se toma el valor res.headers.host y lo pasamos a la vista, quedando el archivo routes/index.js de la siguiente forma:

```
router.get('/', function(req, res) {  
  var host = req.headers.host;  
  
  res.render('index', { title: 'Express', host: host });  
});
```

Y se añade la línea para conectarse al socket:

```
<script>  
  var socket = io.connect('http://<%= host %>');  
</script>
```

A nivel de servidor se añaden varias líneas después de declarar la variable server:

```
var io = require('socket.io')(server);

io.on('connection', function (socket) {

  socket.on('recieve_from_server', function (data) {

    console.log(data);

  });

});
```

Lo único que hace este programa es escuchar si le llega una variable llamada “ recieve_from_server” y mostrarla por pantalla.

Es muy cómodo que con un sólo programa corriendo como demonio se puedan recibir y emitir comunicaciones socket y además hacer de servidor http.

En este paso se añade al archivo index.js de la vista la librería JQuery y se envían datos al servidor mediante un formulario. Este es el ejs completo:

```
<!DOCTYPE html>

<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
    <script src="javascripts/jquery-2.1.1.min.js"></script>
    <script src="javascripts/socket.io.js"></script>
  </head>
  <script>
    var socket = io.connect('http://<%= host %>');
    socket.on('disconnect', function (data) {
      //console.log("disconnect: "+data);
      setTimeout(function(){ location.reload(); }, 900);
    });
  </script>
</html>
```

Node JS y el Puerto Serial

```
    });
}

socket.on('from_server', function (data) {
    console.log(data.body);
    $('#result').text( data.body );
});

function sendToServer () {
    var text=$('#text').val();
    if (text!='') {
        socket.emit('from_client', { my: text});
    }
    $('#text').val("");
}
}
```

```
</head>
<head>
<h1><%= title %></h1>
<p>Bienvenido a <%= host %></p>
```

```
<p>
    <input type="text" name="text" id="text">
<head>
<p>
    <input type="button" name="send" value="send" onclick="javascritp:
sendToServer();">
<head>
{
<h4>
    <span id="result"></span>
```

</h4>

</head>

</html>

El resultado:

8.4 // Node JS trabajando con Serial

Para este ejercicio se carga en el Arduino un programa sencillo:

```

int analogPin3 = 3;      //
int analogPin3 = 3;      //
int analogPin5 = 5;      /
int val3 = 0;            //
int val4 = 0;            //
int val4 = 0;            //
void setup()
{
    Serial.begin(9600);   // setup serial

```

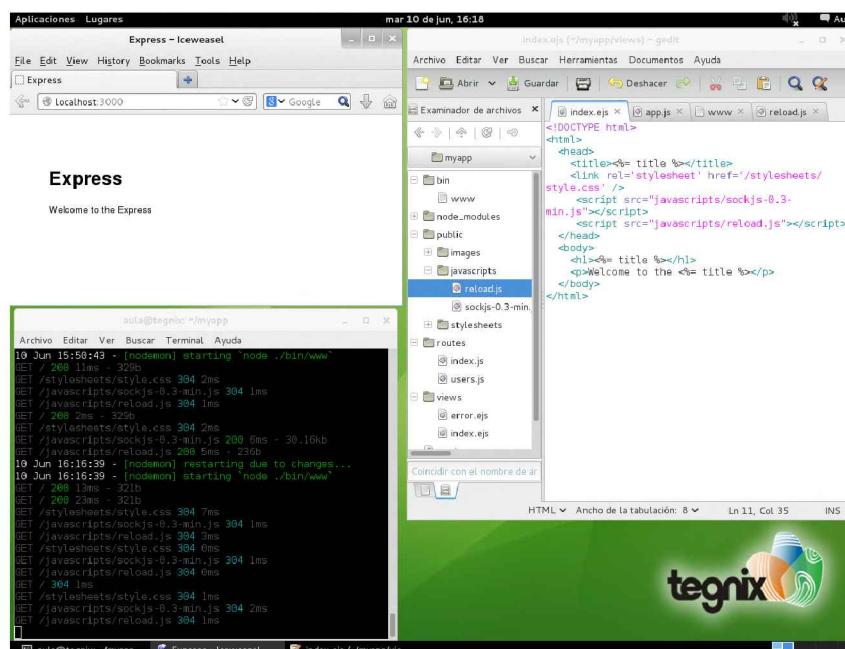


Figura 8.2: Captura pantalla Sockets

Node JS y el Puerto Serial

```
}
```

```
void loop()
```

```
{
```

```
    val3 = analogRead(analogPin3); // read the input pin
```

```
    val3 = analogRead(analogPin3); // read the input pin
```

```
    val3 = analogRead(analogPin3); // read the input pin
```

```
    Serial.print(val3);
```

```
    Serial.print(",");
```

```
    Serial.print(val3);
```

```
    Serial.print(",");
```

```
    Serial.print(val3);
```

```
    Serial.println(); // debug value
```

```
    delay(2000);
```

```
}
```

Lo que hace es enviar 3 valores por el puerto serial que recoge de entradas analógicas separados por comas.

Existen diferentes librerías de Node que leen el puerto serial. La más completa y actualizada que hay en estos momentos es esta:

<https://github.com/voodootikigod/node-serialport>

Se instala con npm con el comando “npm install serialport”.

En este ejemplo se hace un programa con Node JS que sirva como servidor y a parte una página web que recoja los valores enviados.

El archivo de dependencias package.json es este:

```
{
```

```
  "name": "testing_serial",
```

```
  "version": "0.0.1",
```

```
  "private": true,
```

```
  "scripts": {
```

```
    "start": "nodemon -e js,css,ejs app.js"
```

```
    }
  "dependencies": {
    "serialport": "",
    "socket.io": "",
    "socket.io-client": ""
  }
}
```

Este sería el archivo app.js:

```
var app = require('http').createServer(handler)
var io = require('socket.io')(app);
var fs = require('fs');

app.listen(3000);

function handler (req, res) {
  fs.readFile(__dirname + '/web/socket.html',
    function (err, data) {
      if (err) {
        res.writeHead(500);
        return res.end('Error loading index.html');
      };
      res.writeHead(200);
      res.end(data);
    });
}

var com = require("serialport");
//var port="/dev/ttyUSB0";
var port="/dev/pts/6";
var serialPort = new com.SerialPort(port, {
  //baudrate: 9600,
  //parser: com.parsers.readline('\r\n')
});
```

Node JS y el Puerto Serial

```
serialPort.on('open',function() {  
    console.log('Port open');  
});  
  
serialPort.on('data', function(data) {  
    var recieived=data.toString();  
    recieived=recieived.replace(/(\r\n|\n|\r)/gm,"");  
    console.log(recieived);  
    var analogread = recieived.split(',');  
    io.emit('from_server', { analogread: analogread });  
});
```

Se crea una carpeta llamada web con una subcarpeta donde se guardan los archivos necesarios de javascript: jquery-2.1.1.min.js y socket.io.js.

Así quedaría el archivo final de la aplicación web:

```
<!DOCTYPE html>  
  
<html>  
  <head>  
    <meta charset="utf-8">  
    <title>Testing Node JS Serial</title>  
    <style>  
      #analogread0, #analogread1, #analogread2 {  
        display: block;  
        width: 30px;  
        height: 20px;  
        margin: 5px;  
        padding: 5px;  
        text-align: right;  
        color: white;  
      }  
      #analogread0 {
```

```
background: red;  
}  
  
#analogread0 {  
background: green;  
}  
  
#analogread2 {  
background: blue;  
}  
  
<script src="/js/jquery-2.1.1.min.js"></script>  
<script src="/js/socket.io.js"></script>  
<script>  
var socket = io.connect('http://192.168.0.13:3000');  
var min_width=30;  
socket.on('disconnect', function (data) {  
    console.log("disconnect: "+data);  
    setTimeout(function(){ location.reload(); }, 900);  
});  
socket.on('from_server', function (data) {  
    for (i=0; i<data.analogread.length;i++) {  
        //console.log(data.analogread[i]);  
        $("#analogread"+i).text(data.analogread[i]);  
        if (data.analogread[i]<min_width) {  
            data.analogread[i]=min_width;  
        }  
        $("#analogread"+i).width(data.analogread[i]);  
    }  
});
```

Node JS y el Puerto Serial

```
});  
</script>  
</head>  
<head>  
<h1>Read</h1>  
<div id="analogread0">0</div>  
<div id="analogread1">0</div>  
<div id="analogread2">0</div>  
</head>  
</html>
```

El programa en funcionamiento:

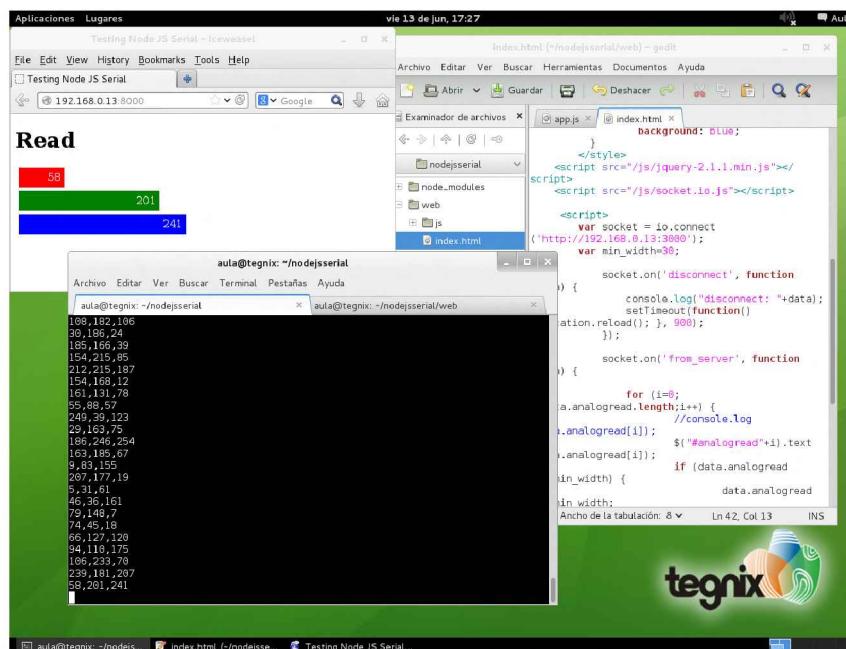


Figura 8.3: Captura pantalla Puerto Serial y Node JS

Capítulo 9

Aplicación Móvil

9.1 // Introducción

Si se quiere hacer un aplicación móvil se pueden optar por dos opciones:

- 1) Decantarse por una plataforma y aprender su lenguaje. Por ejemplo aprender Android.
- 2) Crear una aplicación a partir de código HTML.

Lo normal es elegir la primera opción ya que las aplicaciones nativas tienen más opciones y suelen ser más eficientes. En este caso podemos optar por la segunda opción ya que el programa es muy sencillo y se pueden aprovechar las conexiones socket.

Apache Cordova es una plataforma para construir aplicaciones móviles usando HTML, CSS y Javascript. Es muy conocida ya que es usada por Phonegap, un producto de Adobe, que incluye la compilación “automática” a casi todas las plataformas de móvil existentes.

Existen multitud de frameworks específicos para construir las llamadas apps. Por su facilidad de uso se escoge para este curso Ion Framework, cuya base es Cordova y utiliza como librería de Javascript Angular JS.

9.2 // Angular JS

Como define la Wikipedia.

“AngularJS es un framework de JavaScript de código abierto, mantenido por Google, que ayuda con la gestión de lo que se conoce como aplicaciones de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles.”

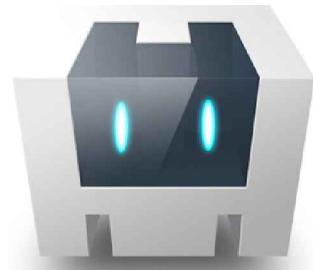


Figura 9.1: Phonegap Logo

Realmente se definen como MVW (Model View Whatever) Modelo Vista “Lo que quieras”, ya que realmente es difícil hablar de un Controlador estricto en Angular JS, aunque este tema entraría en el ámbito de otro curso.

9.2.1 // Hola Mundo con Angular

Con esta estructura se puede empezar un proyecto de Angular.

— index.html

```
  └── js
      ├── angular.min.js
      └── app.js
```



En el index.html se llaman a los archivos necesarios:

```
<script src="/js/angular.min.js"></script>
<script src="/js/app.js"></script>
```

Figura 9.2: Angular JS Logo

Que son la librería de Angular JS y el archivo de aplicación.

Para llamar a Angular JS se necesitan dos pasos, primero en app.js

```
var app = angular.module('myapp', []);
```

Donde ‘myapp’ es el nombre de la aplicación. Ahora en el HTML llamamos a angular.

```
ng-app="myapp".
```

El archivo index.html completo y comentado:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>First test with angular</title>
    <script src="/js/angular.min.js"></script>
    <script src="/js/app.js"></script>
  </head>
```

```
<!-- Muy importante poner el nombre de la aplicación y en controlador
el nombre del controlador. Creamos un alias para que nos sea más fácil.
No es obligatorio que vaya aquí puede ir en cualquier etiqueta -->
```

```
<body ng-app="myapp" ng-controller="AppCtrl as store">
```

Aplicación Móvil

```
</html>

    <!-- Así se unen cadenas de texto -->

        <p>{{“Hola”+”, Tegnix”}}</p>

</html>

    <!-- Operaciones matemáticas -->

        <p>{{18-11}}</p>

</html>

    <!-- Mostramos la variable definida en el contralador -->

        <p>The number is {{store.number}}</p>

</html>

    <!-- Con ng-model se definen variables de formularios -->

        <input name="text" type="text" size=80 ng-model="form.text">

</html>

    <!-- Recogemos el valor del campo input -->

        <h1>You are putting this in the input text “{{form.text}}”</h1>

</html>

        <style>

            <!-- Recorremos con ng-repeat el array people
            y si no pagó la cuota se muestra con ng-show -->

                <li ng-repeat="person in store.people" ng-show="!person.paidQuota">
                    {{person.name}} {{person.paidQuota}}
                </li>
        <style>

</html>

        </head>

</html>

El archivo app.js:

var app = angular.module('myapp', []);

var people=[

    {

        name: ‘Juan’,
```

```
        surname: 'Perro',
        age: 22,
        specialities: [ 'linux', 'debian' ],
        paidQuota: true
    },
}

{
    name: 'Eduardo',
    surname: 'Perro',
    age: 22,
    specialities: [ 'linux', 'windows' ],
    paidQuota: false
},
];


```

```
app.controller('AppCtrl', function() {
    this.number=18;
    this.people=people;
});
```

Se pone en marcha un servidor web con el comando

```
python -m SimpleHTTPServer
```

9.2.2 // Chat con Angular JS y Express JS

Con este ejemplo se profundiza un poco más en Angular JS y con unas pocas líneas se crea un servidor de Chat. Esta sería la parte del Socket.io:

```
var io = require('socket.io')(server);
io.on('connection', function (socket) {
    socket.emit('init', { server: "You are connected to server" });
    socket.on('from_client', function (data) {
```

Aplicación Móvil

```
    console.log(data);
    socket.broadcast.emit('from_server', data);
},
});
```

Este script cuando un cliente se conecta envía una cadena de texto.

Por otro lado al recibir datos desde un cliente lo emite a todos los usuarios conectados ¡Menos al que lo envía! Es la gran ventaja de la orden broadcast.

En Angular añadimos dos nuevos archivos uno para los controladores y otro para los servicios.

Desde los servicios se “inyectan” las funciones necesarias de socket:

```
angular.module('starter.services', [])
.factory('socket', function ($rootScope) {
  var socket = io.connect("http://192.168.0.13:3000");
  return {
    on: function (eventName, callback) {
      socket.on(eventName, function () {
        var args = arguments;
        $rootScope.$apply(function () {
          callback.apply(socket, args);
        });
      });
    },
    emit: function (eventName, data, callback) {
      socket.emit(eventName, data, function () {
        var args = arguments;
        $rootScope.$apply(function () {
          if (callback) {
            callback.apply(socket, args);
          }
        });
      });
    }
  };
});
```

```

    })
},
{
});

```

Solo se hace un controlador llamado AppCtrl:

```

angular.module('starter.controllers', [])
.controller('AppCtrl', function($scope, socket) {
  socket.on('init', function (data) {
    $scope.server = data.server;
  });
  socket.on('from_server', function (message) {
    $scope.messages.push(message);
  });
  //some vars
  $scope.glued = true;
  $scope.messages = [];
  $scope.error="";
  $scope.glued = true;
  $scope.message="";
  $scope.sendMessage = function() {
    if ($scope.your_name=='') {
      $scope.error="Please choose a name";
    } else {
      $scope.error="";
    });
    if ($scope.message!='' && $scope.your_name!='') {
      socket.emit('from_client', { your_name: $scope.your_name, mes-

```

```
sage: $scope.message });

        $scope.messages.push({
            your_name: $scope.your_name,
            message: $scope.message
        });

        $scope.message = '';
    }

},
()
```

Para poder usar una caja de chat donde automáticamente se va ajustando el texto al fondo usamos una librería llamada “luegg” que se puede descargar desde aquí <https://raw.githubusercontent.com/Luegg/angularjs-scroll-glue/master/src/scrollglue.js>. Se llama desde app.js:

```
angular.module('starter', ['luegg.directives', 'starter.controllers', 'starter.services']);
```

Para entender el funcionamiento de este controlador: al darle al botón enviar (o pulsar la tecla “intro”) si hay escrito un nombre envía al servidor el nombre de quien lo envía y el mensaje. Así mismo en el momento que recibe un mensaje lo añade a la variable “messages” y Angular JS se encarga de mostrarlo en pantalla.

Se aprovecha el archivo index.js que hay por defecto en la carpeta view al instalar Express JS.

```
<!DOCTYPE html>

<html>
    <head>
        <title><%= title %></title>
        <link rel='stylesheet' href='/stylesheets/style.css' />
        <script src="/javascripts/socket.io.js"></script>
        <script src="/javascripts/angular.min.js"></script>
        <script src="/javascripts/scrollglue.js"></script>
        <script src="/javascripts/app.js"></script>
        <script src="/javascripts/services.js"></script>
        <script src="/javascripts/angular.min.js"></script>
```

```
</head>

<body ng-app="starter" ng-controller="AppCtrl">

    <h1><%= title %></h1>

    <h2 ng-show="server">
        {{server}}
    </h2>

    <input type="text" placeholder="Your name . . ." ng-model="your_name">

    <span class="error" ng-show="error">
        {{error}}
    </span>

    <form ng-submit="sendMessage()">
        Message: <input size="60" ng-model="message">
        <input type="submit" value="Send">
    </form>

    <div scroll-glue>
    >;
        <li ng-repeat="message in messages">{{message.your_name}}: {{message.message}}</li>
        age: 22,
    </li>
    </div>
</head>

</html>

El CSS:

[scroll-glue] {
    height: 240px;
    width: 500px;
    padding: 15px;
    margin: 10px;
    overflow-y: scroll;
    border: 1px solid black;
}
```

Aplicación Móvil

El resultado final:

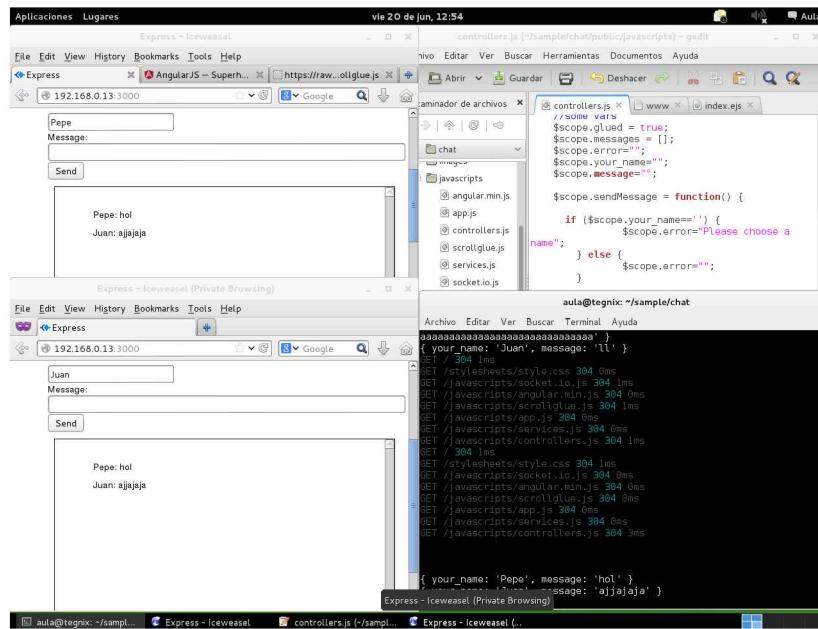


Figura 9.3: Resultado final chat

9.3 // Hola mundo con Ionic Framework

De todos los frameworks que existen para realizar Apps móviles se elige Ionic por las siguientes razones:

- Su base es Cordova
- Usa Angular JS
- Dispone de un CSS que cubre todas las necesidades de botones, sliders, menús, ...
- Facilidad de uso



Figura 9.4: Ionic Logo

Se instalan los dos paquetes de una sola vez Cordova e Ionic:

```
npm -g install cordova ionic
```

Una vez instalado para empezar una aplicación:

```
ionic start myionic sidemenu
```

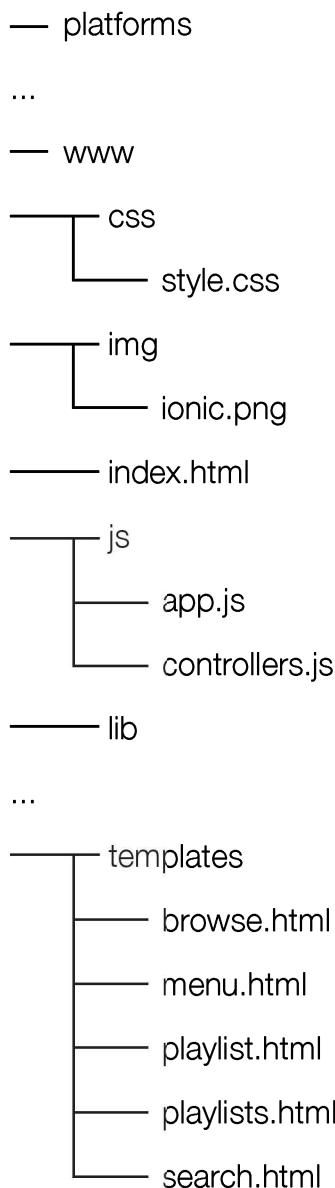
Las opciones son “blank” (página en blanco), “tabs” y “sidemenu” (menú lateral).

Se escoge la tercera por ser la más completa.

Dentro de la estructura estos son los archivos más importantes:

...

Capítulo 9



En la carpeta “platforms” se guardan los archivos para los diferentes dispositivos: Android, iOS, ...

En “www” están los archivos que realmente modificamos. Se pueden ver dentro de la carpeta “js” los archivos programados en Angular JS.

Dentro de “lib” se encuentran todas las librerías necesarias de javascript y CSS.

¿Cómo se puede probar la aplicación? Muy fácil:

```
ionic serve
```

Dentro de “app.js” se puede ver como se llaman a los “templates” y como se utilizan las rutas:

Aplicación Móvil

Resultado final:

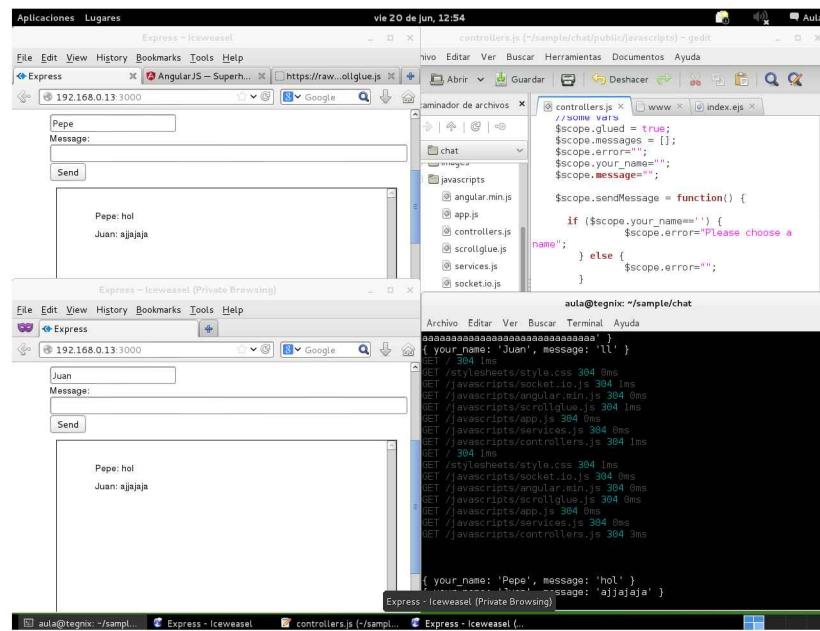


Figura 9.5: Ionic App en acción

10.1 // Introducción

Para explicar qué es Raspberry Pi, citamos textualmente su entrada en la Wikipedia:

“Raspberry Pi es una placa computadora (SBC) de bajo coste desarrollada en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas.

El diseño incluye un System-on-a-chip Broadcom BCM2835, que contiene un procesador central (CPU) ARM1176JZF-S a 700 MHz (el firmware incluye unos modos “Turbo” para que el usuario pueda hacerle overclock de hasta 1 GHz sin perder la garantía), un procesador gráfico (GPU) VideoCore IV, y 512 MB de memoria RAM aunque originalmente al ser lanzado eran 256 MB. El diseño no incluye un disco duro o una unidad de estado sólido, ya que usa una tarjeta SD para el almacenamiento permanente; tampoco incluye fuente de alimentación o carcasa. El modelo B se vende a 35 \$ y el modelo A a 25 \$.”

La definición en si misma ya responde a la pregunta de porqué Raspberry Pi. Basicamente por que es el medio ideal para aprendizaje debido a su bajo coste y versatilidad. Existen varias distros Linux preparadas para ser instaladas en la SD, y eso nos otorga una gran potencia y flexibilidad para nuestros proyectos.

10.2 // Montando el hardware:

Una vez que recibimos nuestra flamante Raspberry Pi, comprobamos que todavía nos faltan varios componentes para ponerla en marcha, si bien su precio no influirá demasiado en el coste final:

Por una parte, necesitamos una fuente de alimentación. Es imprescindible que tenga un mínimo de calidad ya que es uno de los componentes primordiales, y de ella depende que el sistema funcione de manera estable sin cuelgues. El fabricante recomienda que entrene 5v. y un mínimo de 700 mA. Además debe poseer un conector microusb.

Tarjeta SD o micro SD (con su correspondiente adaptador). Mínimo de 4GB de capacidad

y sobre todo, por motivos de rendimiento, que sea de clase 4 o superior. Mi experiencia me dice que son con tarjetas de poca capacidad (4GB sobre todo), donde se obtienen los mejores rendimientos.



Figura 10.1: Raspberry PI

Esto es todo el material que necesitamos para tener nuestra Raspberry Pi lista para funcionar.

Nota: durante el proceso de instalación del sistema operativo, se necesitan además un teclado y ratón usb, así como un cable de vídeo (hdmi o vídeo compuesto con conectores rca).

10.3 // Instalando el Sistema Operativo:

Existen varias distros de Linux para Raspberry Pi, incluso tarjetas de memoria SD con el S.O preinstalado. Nosotros elegimos la distro Raspbian por ser fácil de instalar y optimizada para la Raspberry Pi.

Nota: la tarjeta SD no tiene el rendimiento de un disco duro, por este motivo, dependiendo de la cantidad de paquetes seleccionados, la instalación se puede demorar bastantes minutos.

10.3.1 // Paso 1 (preparación de la tarjeta SD):

En nuestro PC con el SO GNU/Linux, instalamos (en caso de no tenerlo), el editor de particiones Gparted. Con este software procedemos a formatear toda la tarjeta SD con el sistema de ficheros FAT32.

10.3.2 // Paso 2 (descarga de los archivos de instalación):

Descargamos el fichero de la url (http://archive.raspbian.org/installer/rpi_installer_08-19-12.zip), lo descomprimimos y copiamos los ficheros de instalación resultantes dentro de la tarjeta SD.

10.3.3 // Paso 3 (conectándolo todo):

Desmontamos la SD de nuestro PC y la insertamos en la Raspberry Pi junto a un teclado y ratón usb, le conectamos también un cable de red con salida a Internet (imprescindible que el dispositivo gateway esté configurado para ofrecer direcciones IP por DHCP). También conectamos el monitor, bien a través de la interfaz HDMI o por el conector RCA (nos dará una calidad más pobre, pero sólo se usará durante el proceso de instalación). Por último conectamos el adaptador de corriente al conector micro usb.

Nota: necesitaremos adicionalmente, un cable HDMI o de vídeo compuesto con terminales RCA dependiendo del tipo de monitor empleado.



Figura 10.2: Cable HDMI



Figura 10.3: Cable Vídeo compuesto con conectores RCA

10.3.4 // Paso 4 (instalación propiamente dicha):

Una vez que tenemos todo conectado, encendemos la Raspberry Pi , y si todo va bien, veremos en el monitor como carga un sistema básico que iniciará el instalador. Éste nos guiará por todo el proceso de instalación del sistema operativo. A continuación enumero las diferentes preguntas que nos hará y sus correspondientes respuestas:

- La primera pantalla nos invita a seleccionar un idioma para el sistema operativo, nosotros escogemos la opción “Spanish”.
- Seleccione su ubicación: “España”.

- Mapa de teclado a usar: “Español”.

Ahora intentará detectar el hardware de red y lo configurará con los parámetros que nuestra Gateway le suministrará a través del protocolo DHCP. (dirección IP, puerta de enlace, servidores de DNS).

- Seleccione el nombre de la máquina: El que queramos.
- Nombre de dominio: como la vamos a usar en una red doméstica, en principio dejamos la opción por defecto.
- País de la réplica de Debian: “Introducir información manualmente”
- Nombre del servidor de la réplica de Debian: “mirrordirector.raspbian.org”
- Directorio de la réplica de Debian: “/raspbian/”
- Información de proxy http: lo dejamos en blanco.

En la siguiente pantalla nos saldrá un error que no se han encontrado módulos del núcleo. Le decimos que Sí a continuar la instalación sin cargar módulos del núcleo. (Se tratan de módulos de sistemas de ficheros casi en desuso, crypto, etc., los cuales no nos deberían de dar problemas ya que muchos ya están compilados dentro del kernel y otros ya no se usan).

- Definimos una contraseña para el superusuario (root): la que queramos.
- Creamos un usuario normal: Por ejemplo “pi” (tanto en nombre completo como de usuario).
- Contraseña para el usuario pi: la que queramos.
- Seleccionamos la ubicación para la zona horaria: en nuestro caso “Península”.
- Ahora se nos muestra un resumen de la tabla de particiones (creada automáticamente). La dejamos como está y seleccionamos “Finalizar el particionado y escribir cambios en el disco”.

NOTA: básicamente crea una partición FAT32 donde ubica los ficheros de inicio del sistema, una partición de intercambio (SWAP o el equivalente a la memoria virtual en Windows), y por último una partición raíz EXT3 donde se ubicarán los ficheros principales del sistema.

- Desea escribir los cambios en los discos: “Sí”

Ahora empezará el proceso de instalación y configuración del sistema, que dependiendo del tipo de tarjeta SD, le llevará un tiempo.

■ Al final del proceso se nos mostrará un aviso “No se encontró un núcleo instalable en las fuentes APT definidas”, a esto le decimos que Sí a continuar sin instalar ningún núcleo.

■ El siguiente paso es seleccionar si queremos participar en la encuesta sobre el uso de paquetes (para generar estadísticas sobre qué paquetes son los más usados): la respuesta la dejamos a la decisión de cada uno.

■ Elegimos el software a instalar: solamente seleccionamos el servidor SSH y “Utilidades estándar del sistema”.

■ A la pantalla de “no se ha instalado ningún cargador de arranque”, elegimos “continuar”.

Completado el proceso, el sistema se reiniciará y ya podremos ingresar en él mediante las credenciales que configuramos en el apartado de instalación.

Una vez instalado el sistema operativo en la Raspberry Pi, ya no necesitamos conectarla a un monitor, podemos trabajar a través de otro pc que esté en la misma subred por ssh.

Como la dejamos configurada para obtener los parámetros de red por DHCP, todavía antes de desconectarle el monitor debemos averiguar qué ip le ha otorgado la gateway:

■ Como root tecleamos la orden “ifconfig”, que entre otros parámetros, nos indicará qué ip tiene asignada nuestra Raspberry Pi.

■ Ahora ya desde el pc de trabajo podemos conectarnos a ella mediante ssh con el siguiente comando: “ssh pi@ip_asignada”. Pi es el usuario que creamos en su momento (a mayores del superusuario o root).

■ Seguidamente nos pedirá la clave del usuario “pi”, y ya estamos logueados en el sistema Linux.

De todas formas, sería bastante engorroso cada vez que queramos conectarnos a través de ssh, averiguar qué ip tiene ahora asignada, así que vamos a configurarle una ip estática, además de todos los parámetros de red que necesita para conectarse a Internet:

■ Para hacer cambios en el sistema tendremos que loguearnos como root. Tecleamos “su” seguido de la password que asignamos al usuario root en el momento de la instalación.

- Editamos el fichero de configuración de red “nano /etc/network/interfaces”. Tendrá que quedar como esto:

```
# The primary network interface  
  
auto eth0  
  
iface eth0 inet static  
  
address 192.168.1.100  
  
netmask 255.255.255.0  
  
gateway 192.168.1.1
```

Nota: el apartado “address” y “gateway” se configurarán según los parámetros de vuestra red.

- Guardamos los cambios con **CRTL+o** y salimos con **CRTL+x**
- Configuramos los servidores de dns “nano /etc/resolv.conf”. Yo he puesto los de Google: (primario 8.8.8.8 y secundario 8.8.4.4).
- Reiniciamos el servicio de red “service networking restart”. Perderemos la conexión ya que hemos cambiado su dirección ip, con lo cual tendremos que volver a conectarnos.

Procedemos también a instalar con el gestor de paquetes de Debian las utilidades para poder compilar software. Estas utilidades entre otros, incluyen el famoso compilador de C de GNU gcc:

```
“apt-get install build-essential”
```

10.4 // Control de los GPIO

Los puertos GPIO (General Purpose I/O) de la Raspberry Pi, son una especie de interfaces de bajo nivel diseñados para conectarla directamente con otros chips o módulos. Estos pines usan un nivel de voltaje de 3 voltios, y no aceptan niveles de salida de 5 voltios, como en el caso de Arduino.

Los puertos GPIO se pueden configurar como de entrada o salida y se pueden controlar por software. En la siguiente imagen se puede apreciar el esquema de los puertos GPIO de la Raspberry Pi.

En el siguiente ejercicio vamos a usar uno de los puertos GPIO de la Raspberry Pi (17), para encender y apagar un led.

Este es el diagrama de conexionado:



Figura 10.4: Esquema puertos GPIO

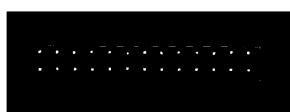


Figura 10.5: Puertos físicos GPIO

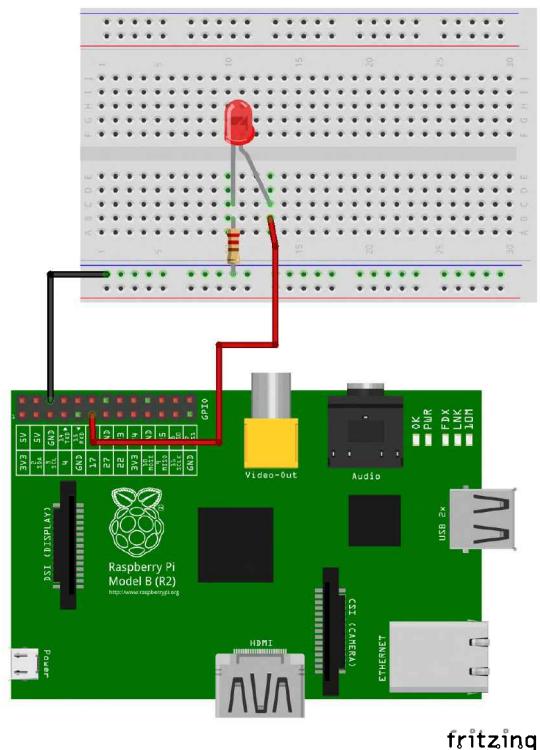


Figura 10.6: Diagrama conexiones

Para Linux, todo es un archivo (incluidos los dispositivos), de esta manera los puertos GPIO los trata como tal; así podemos utilizar los comandos básicos de Linux para interactuar con ellos. Vamos a ver cómo:

Lo primero es activar un pin, ya que hasta ahora todos están desactivados por defecto. Activaremos el GPIO 17 con el siguiente comando:

```
echo 17 > /sys/class/gpio/export
```

Esto crea un fichero con una estructura GPIO dentro del subdirectorio gpio. Se trata de un acceso al GPIO17.

Nota: para ejecutar los comandos es necesario iniciar sesión como root.

Como vamos a conectar un led en el GPIO 17, éste va a ser un pin de salida, y se lo decimos a la Raspberry Pi con el siguiente comando:

```
echo out > /sys/class/gpio/gpio17/direction
```

Raspberry Pi

Por último tenemos que darle valores de encendido (1) o apagado (0), y lo haremos de la siguiente manera:

```
echo 1 > /sys/class/gpio/gpio17/value (led encendido)
```

```
echo 0 > /sys/class/gpio/gpio17/value (led apagado)
```

Subir sketches a Arduino desde RPi

En este apartado vamos a explicar brevemente cómo subir un programa a Arduino mediante la línea de comandos. La idea es que no tengamos que usar el IDE de Arduino para poder subir los sketches, y podamos hacerlo mediante la línea de comandos una vez conectados a la Raspberry Pi mediante el protocolo ssh.

La herramienta que usaremos es inotool (<http://www.inotool.org>).

Lo primero que haremos será instalar las dependencias:

```
apt-get install python-setuptools
```

```
apt-get install python-configobj
```

```
apt-get install python-jinja2
```

```
apt-get install python-jinja2
```

```
apt-get install picocom (para el monitor serial).
```

```
apt-get install python-pip (utilidad para gestionar módulos de Python).
```

```
apt-get install arduino-core
```

Ahora instalaremos la utilidad “git”. Git es un software de control de versiones, y lo necesitamos para descargar el código fuente:

```
apt-get install git
```

Obtenemos y compilamos el código fuente:

[git clone git://github.com/amperka/ino.git](https://github.com/amperka/ino.git)

```
cd /ino
```

```
make install
```

Instalamos el módulo “glob2” para Python:

```
pip install glob2
```

Ahora que ya tenemos todo el software necesario, vamos a comenzar un proyecto:

```
mkdir blink (creamos un directorio de prueba).
```

```
Cd blink
```

```
ino init (crea la estructura del proyecto).
```

Dentro del subdirectorio **src/** del proyecto ya tenemos un sketch de ejemplo; pero aquí será donde pongamos nosotros el que queramos. Podemos usar un sketch de algún ejercicio de Arduino, sólo tenemos que crear un fichero vacío con extensión .ino y pegar dentro el código que queramos.

Nota: Cuando compilamos y subamos el sketch, la utilidad parte de la base de que tenemos como placa un Arduino Uno; en caso contrario deberemos crear un fichero de configuración con el nombre “ino.ini”, que ubicaremos en la carpeta raíz del proyecto y que contendrá el nombre de la placa y opcionalmente el puerto al que está conectada:

```
[build]
```

```
board-model = mega2560
```

```
[upload]
```

```
board-model = mega2560
```

```
serial-port = /dev/ttyACM1
```

```
[serial]
```

```
serial-port = /dev/ttyACM1
```

Compilamos el proyecto (hay que hacerlo en la carpeta raíz del mismo):

```
ino build
```

Por último, subimos el sketch a la placa Arduino:

```
ino upload
```

Capítulo 11

Ejercicios finales

11.1 // Introducción

En este tema se van a hacer tres ejercicios que cubren todas las posibilidades de comunicación entre un Arduino y Node JS.

11.2 // Enviando texto a un LCD

Este ejemplo tiene como objetivo poder enviar un texto desde una aplicación web a un LCD conectado a un Arduino. En este caso y en los siguientes en vez de trabajar sobre la máquina virtual se hará en una Raspberry Pi.

En primer lugar se monta este esquema sobre la placa de prototipado.

El LCD (Liquid Crystal Display) es un componente muy interesante ya que nos puede ofrecer información de lo que ocurre dentro del Arduino de forma escrita. En este caso se emplea uno de dos líneas aunque en el mercado existen gran variedad de ellos.

En el Arduino debe estar cargado el programa Firmata Standard. Se puede encontrar en **Ejemplos > Firmata > Standard Firmata**.

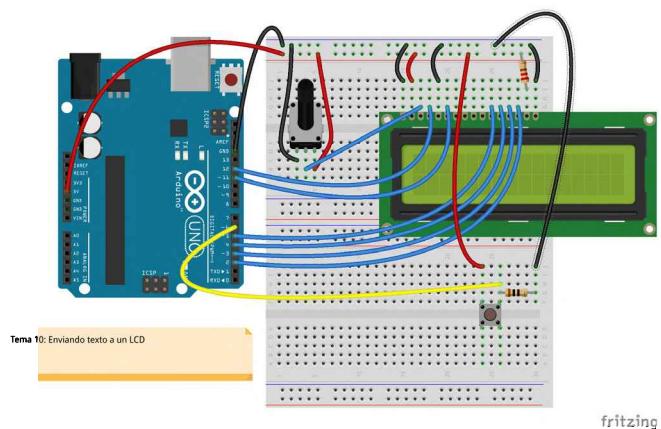


Figura 11.1: Sketch LCD

Se necesita un servidor de sockets que será el encargado de regular el tráfico de información entre la aplicación y el arduino.

Para entender el funcionamiento de Node JS tenemos que saber que es Asíncrono ¿Qué significa esto? Como explica muy bien Raquel Vélez en su artículo “Understanding Async” (<http://rckbt.me/2014/05/understanding-async/>):

Necesito cambiar de vestuario y puedo conseguirlo de las siguientes formas:

De forma Síncrona: salgo de casa voy a una zapatería y compro unos zapatos, después en una tienda compro una camiseta y para terminar voy a otra tienda y me compro unos pantalones.

De forma Asíncrona: entro en una tienda de ropa y compro unos pantalones, al mismo tiempo desde otra web compro unos zapatos (el pedido de los pantalones ya está en proceso) y para terminar compro unas camisetas desde una aplicación móvil. El proceso está en marcha pero no sabemos que nos va a llegar primero.

Trabajar de forma Asíncrona está muy bien porque nos permite aprovechar mejor los recursos, es decir, hacer varias cosas a la vez. Pero hay veces que necesitamos cierto orden, como en este caso, primero nos conectamos a Arduino y después escuchamos llamadas al socket. Si no fuera así tendríamos errores en la aplicación.

Una forma de “forzar” este orden es embebiendo funciones dentro de funciones pero a la hora de leer este código puede ser muy difícil. Un método sencillo es usar la librería “async”.

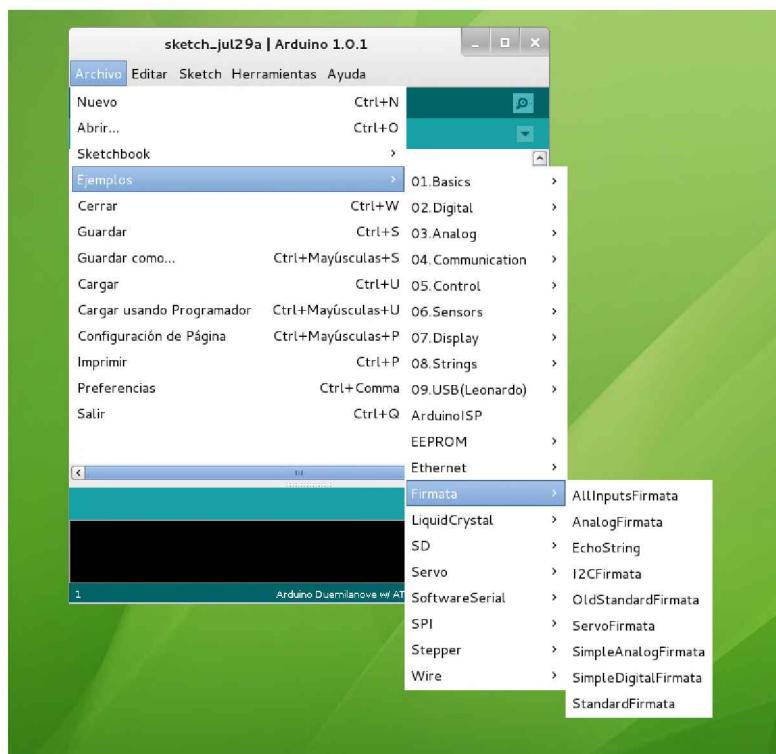


Figura 11.2: Sketch Firmata

Ejercicios finales

Como se puede ver en el ejemplo se llama:

```
async = require("async");
```

Para comunicarnos con el Arduino se usa la librería “Johnny Five” que trae una función incorporada para enviar texto a un LCD.

El código completo:

```
var app = require('http').createServer(handler)
var io = require('socket.io')(app);
var fs = require('fs');

app.listen(8000);
```

```
function handler (req, res) {
  fs.readFile(__dirname + '/index.html',
    function (err, data) {
      if (err) {
        res.writeHead(500);
        return res.end('Error loading index.html');
      }
      res.writeHead(200);
      res.end(data);
    }
  );
}
```

```
async = require("async");
```

```
var five = require("johnny-five"),
  board, lcd;
board = new five.Board();

board.on("ready", function() {
```

```
  async.series([
```

```
function(callback) {  
  //  
  board.on("ready", function() {  
  
    lcd = new five.LCD({  
      // LCD pin name  RS  EN  DB4  DB5  DB6  DB7  
      // Arduino pin # 12  11  5   4   3   2  
      pins: [12, 11, 5, 4, 3, 2],  
      rows: 2,  
      cols: 16  
      // Options:  
      // bitMode: 4 or 8, defaults to 4  
      // lines: number of lines, defaults to 2  
      // dots: matrix dimensions, defaults to "5x8"  
    });  
    //  
    button = new five.Button(6);  
    //  
    button.on("press", function() {  
      callback(null, null);  
    });  
  });  
},  
//  
function(callback) {  
  //  
  io.on('connection', function (socket) {  
    //  
    socket.emit('init', { server: 'Welcome to server' });  
  });  
};
```

Ejercicios finales

```
socket.on('from_client', function (data) {  
});  
lcd.clear().print(data.line1);  
console.log('line1: '+data.line1);  
});  
lcd.cursor(1, 0);  
console.log('line1: '+data.line1);  
lcd.print(data.line2);  
rows: 2,  
});  
  
button.on("down", function() {  
    socket.emit('warning', { data: 1 });  
});  
  
});  
  
callback(null, null);  
},  
  
function(err, results) {  
    console.log("End");  
}  
};
```

Se ha añadido un Switch para ensayar como se reciben cambios en una Entrada Digital. En este caso cuando el botón está pulsado se envía al socket un aviso.

```
button.on("down", function() { ...
```

Por último se usa IonicFramework para la aplicación móvil. Lo más importante en esta aplicación es el controlador que se hace para enviar las líneas y lo que se muestra cuando el botón es pulsado.

Capítulo 11

Archivo www/js/controllers.js:

```
controller('LCDCtrl', function($scope, $stateParams, $timeout, socket) {  
  
    $scope.warning='';  
  
    socket.on('warning', function (data) {  
        $scope.warning = data.data;  
        $timeout(function(){  
            $scope.warning='';  
        }, 2000);  
    });  
  
    $scope.writeLCD = function() {  
  
        if ($scope.writeLCD.line1 && $scope.writeLCD.line2) {  
            console.log("$scope.writeLCD.line1: "+$scope.writeLCD.line1+",  
$scope.writeLCD.line2: "+$scope.writeLCD.line2);  
            socket.emit('from_client', { line1: $scope.writeLCD.line1, line2:  
$scope.writeLCD.line2 });  
        }  
    }  
})
```

En el archivo www/js/services.js hay que decirle donde está el servidor de socket, en este caso la IP de la Raspberry Pi es 192.168.0.104.

```
var socket = io.connect("http://192.168.0.104:8000");
```

En app.js se crea la ruta lcd

```
.state('app.lcd', {  
    url: "/lcd",
```

Ejercicios finales

```
if (err) {  
    'menuContent' :{  
        templateUrl: "templates/lcd.html",  
        controller: 'LCDCtrl'  
    });  
},  
},  
};
```

Que además es la que hay por defecto al entrar en la aplicación:

```
$urlRouterProvider.otherwise('/app/lcd');
```

El ejemplo final:

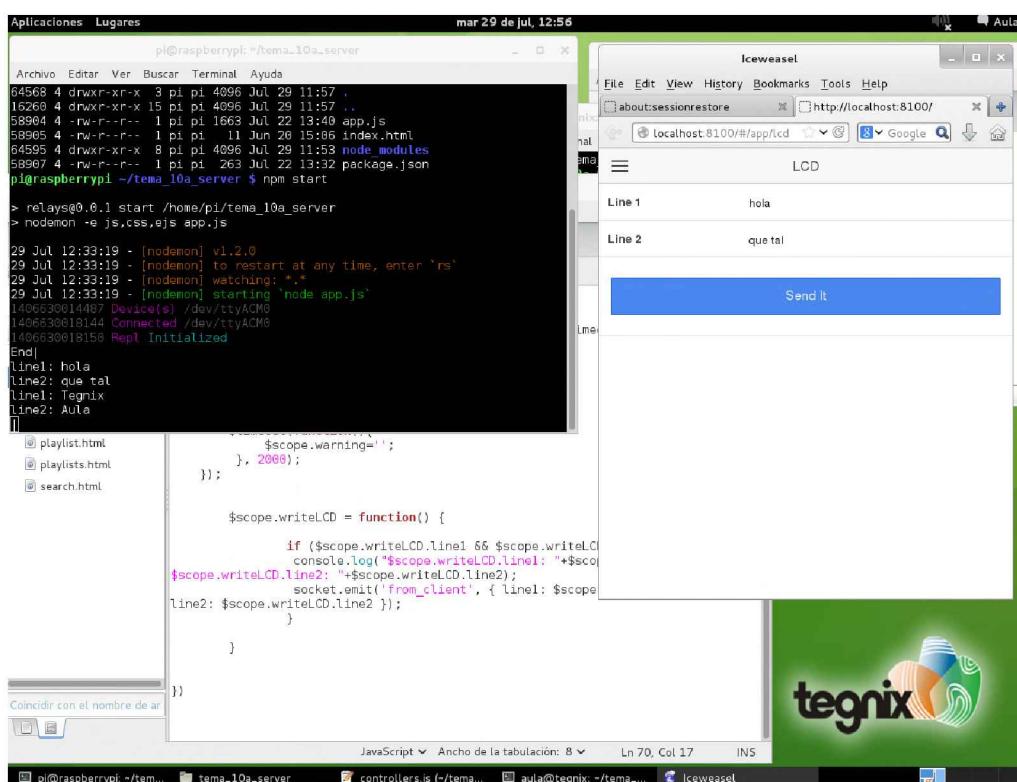


Figura 11.3: Ejemplo final LCD

11.3 // Placa de Relés

Para este ejercicio se usa una placa que se acopla al Arduino que contiene 4 relés. Su nombre es “Relay Shield” y toda la información técnica se puede leer desde http://www.seeedstudio.com/wiki/Relay_Shield_V2.0



Figura 11.4: Placa de relés

Utiliza los puertos digitales 4, 5, 6 y 7 del Arduino con lo que es muy fácil programar.

Este ejemplo se divide en dos partes:

El servidor: en este caso se usa la librería de Node JS “Arduino Firmata”. Es una librería más avanzada que la primera que salió a la luz la llamada “Firmata” a secas. Se puede ver más información desde <https://github.com/shokai/node-arduino-firmata>.

En el Arduino se carga el programa Firmata Standard como en el ejemplo anterior.

En el archivo principal app.js se define un array que servirá para almacenar los estados de los Relés.

```
var relays=[  
    { number: 0, text: "Relay1", checked: false },  
    { number: 0, text: "Relay1", checked: false },  
    { number: 0, text: "Relay1", checked: false },  
    { number: 3, text: "Relay4", checked: false }  
],
```

Cuando la aplicación se conecta al servidor lo primero que se hace es enviar este array:

```
socket.emit('init', relays);
```

Ejercicios finales

Al recibir una orden desde “from_client” la enviamos como “broadcast” a todos los posibles aplicaciones abiertas.

```
socket.broadcast.emit('from_server', data);
```

Así mismo se envía al Arduino la orden y se actualiza la variable “relay”.

```
arduino.digitalWrite(number, data.state);  
relays[data.number].checked = data.state;
```

Por parte del cliente en el archivo www/js/controllers.js:

```
angular.module('starter.controllers', [])  
.controller('AppCtrl', function($scope, socket) {  
  
    socket.on('init', function (data) {  
        $scope.settingsList = data;  
        //console.log(data);  
    }],  
  
    socket.on('from_server', function (data) {  
        //console.log(data)  
        $scope.settingsList[data.number].checked=data.state;  
    }],  
  
    $scope.doLight = function(state, number) {  
        //console.log(state+” +number);  
        socket.emit('from_client', { state: state, number: number });  
    }  
})
```

En cada uno de los apartados se realizan los siguientes cambios:

- En “init”: guardamos el estado de los relés.
- En “from_server” cambiamos el estado del botón. Importante: recordemos que la aplicación que se ejecuta no va a recibir de vuelta esta información ya que el servidor hace un “broadcast” con ella.

- En la función “doLight” que se ejecuta cuando hacemos un click en el botón emitimos al servidor la orden.

Por último en el archivo index.html expresamos toda esta información:

```
<ion-toggle ng-repeat="item in settingsList">  
    ng-model="item.checked"  
    ng-checked="item.checked" ng-click="doLight(item.checked,  
        item.number)">  
        {{ item.text }}  
</ion-toggle>
```

En este ejercicio por un lado se usa un emulador de Android y por el otro el propio servidor de Ionic.

Es importante ejecutar el comando “android” y añadir el “target” 1.9 de API de Android:

Primero se añade la plataforma Android al proyecto:

```
ionic platform add android
```

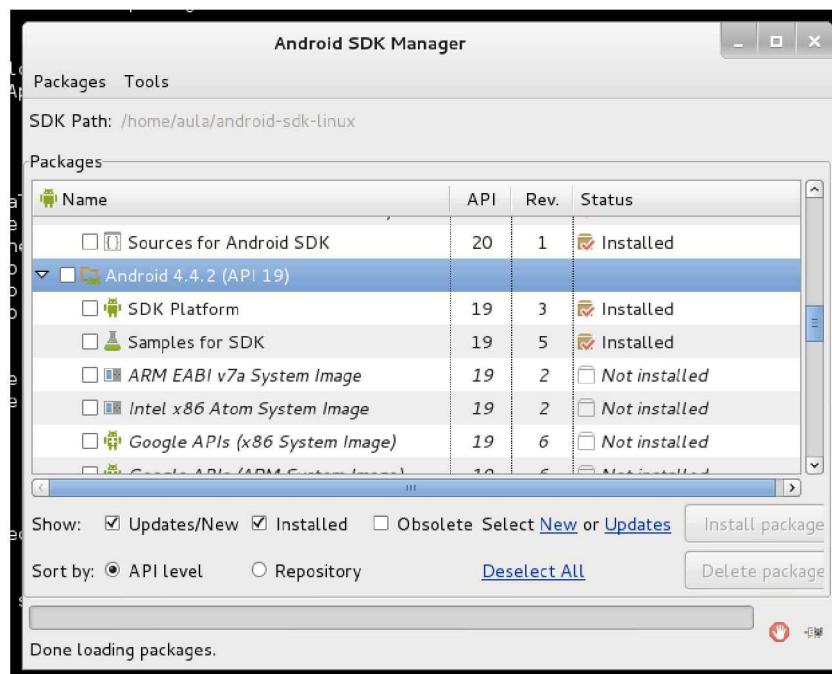


Figura 11.5: Android SDK

Ahora se añade una máquina virtual de Android que se llaman AVD (Android Virtual Device). Para ello desde el SDK se va a “Tools” → “Manage AVDs”. Se hace clic en “Create”. Se elige como ejemplo el “Nexus S”.

Ejercicios finales

Para realizar una emulación:

ionic emulate android

Esta es la pantalla final con el resultado:

En general las máquinas virtuales de Android tienen un funcionamiento bastante pobre. Se recomienda usar un móvil para las pruebas. Para ello solo hace falta activar el “Modo desarrollador”, a partir de la versión 4.2 se hace de la siguiente forma:

- Ir a “Ajustes” en el terminal.
- En el final de la lista entrar en “Información del dispositivo”.
- Dentro del menú de “Información del dispositivo” pulsar 7 veces sobre el campo de “Número de Compilación”.
- Aparecerá en la pantalla el mensaje ¡Ahora eres un Desarrollador!
- Habrá un nuevo menú de Opciones de Desarrollo en los menús de Ajustes.

Se conecta el móvil por USB al ordenador y se ejecuta:

ionic run android

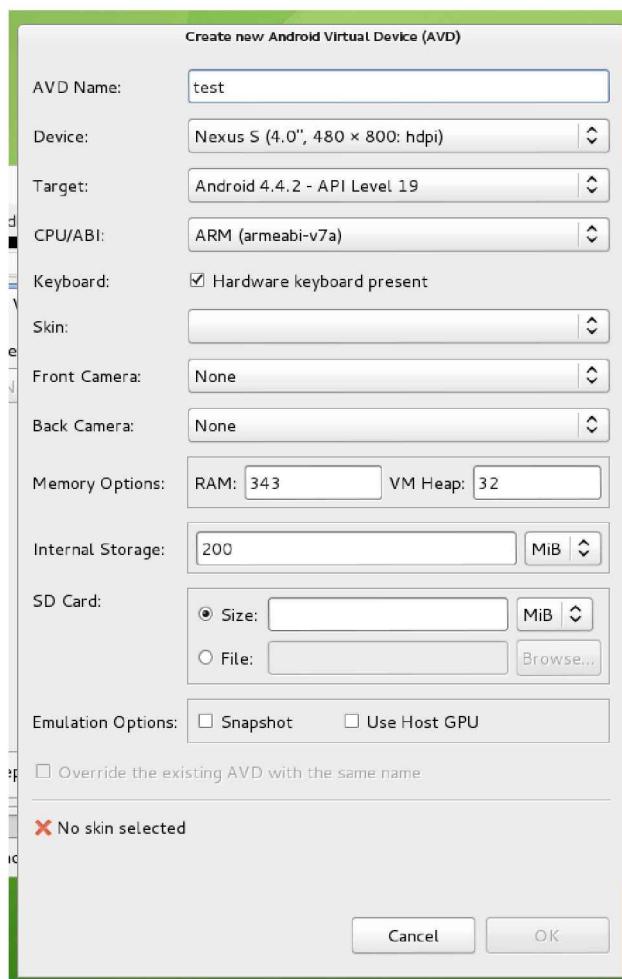


Figura 11.6: Máquina Virtual de Android

Capítulo 11

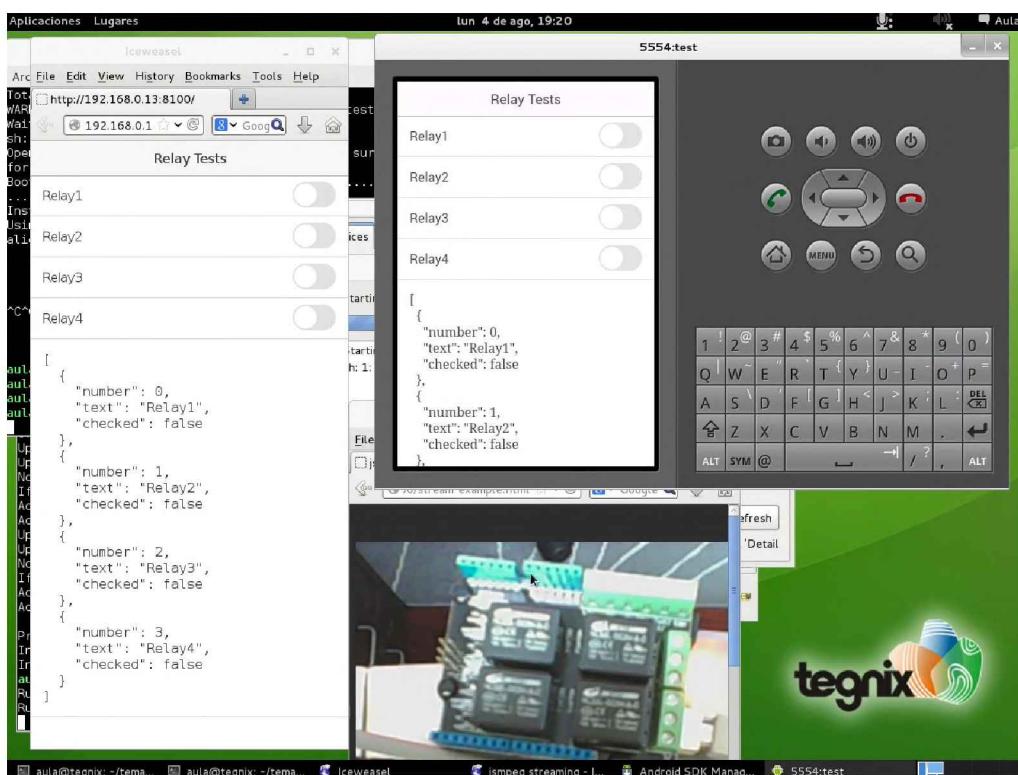


Figura 11.7: Captura pantalla Relés

11.4 // Moviendo un servo

Para finalizar este curso se realiza un ejemplo con un servo. El esquema es el siguiente:

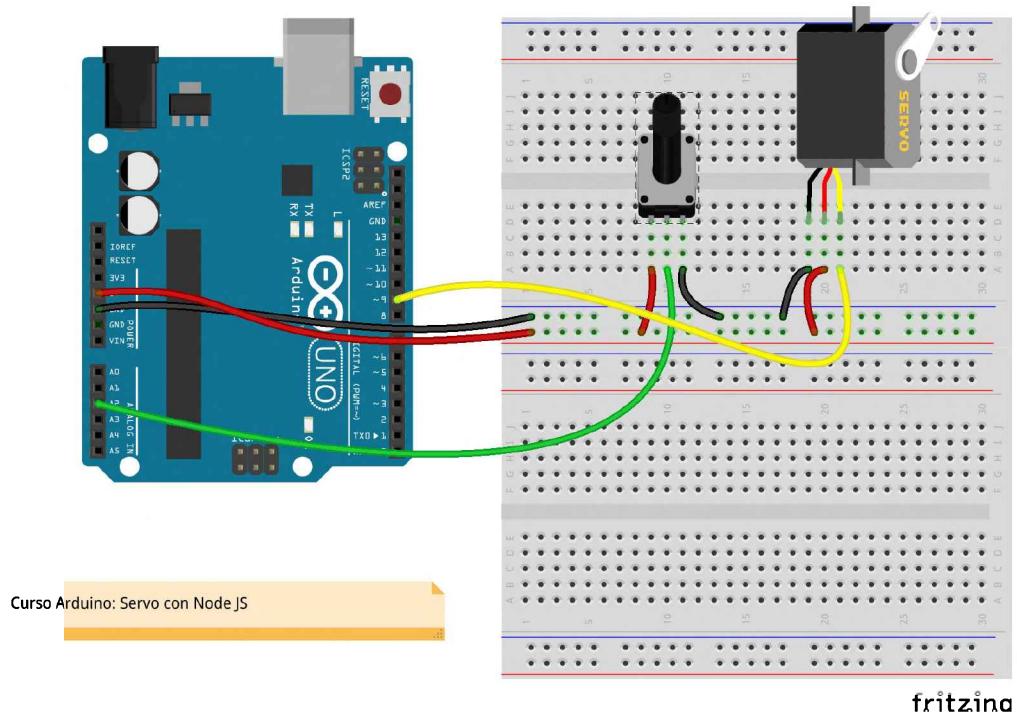


Figura 11.8: Servo Sketch

Se utiliza la librería de node jhonny-five con lo que al igual que en el ejemplo anterior el Arduino lleva cargado el programa Firmata.

El potenciómetro del esquema se va a utilizar para enviar una señal analógica al servidor de Node JS.

Si se analiza el programa de servidor de Node JS:

```
var init_servo=0;  
  
var five = require("johnny-five"),  
    board, potentiometer, servo, scalingRange;  
  
board = new five.Board();  
  
board.on("ready", function() {  
    scalingRange = [0, 170];  
  
    servo = new five.Servo({  
        pin: 9,  
        range: scalingRange  
    });  
});
```

```
});  
potentiometer = new five.Sensor({  
    pin: "A2",  
    freq: 2000  
});  
potentiometer.on("read", function( err, value ) {  
    board.emit('value', value);  
});  
});  
  
io.on('connection', function (socket) {  
    socket.emit('init', { value: init_servo });  
      
    socket.on('from_client', function (data) {  
        //console.log("from_client: "+data.value);  
        init_servo=data.value;  
        servo.to(Math.floor(data.value));  
        socket.broadcast.emit('from_server', data);  
    });  
      
    board.on('value', function(val){  
        console.log("val: "+val);  
        socket.emit('value', {val: val});  
    });  
});
```

Con la variable “ scalingRange” se limita el angulo de giro del servo en este caso entre 0 y 170 grados.

Se define el potenciómetro en la salida analógica “A2” con que se va a leer con una frecuencia de 2000 mili segundos.

Cada vez que se lea se envía la variable “value” con “board.emit” que se recoge más adelante en “board.on('value' ... ”.

Ejercicios finales

Cuando se recibe desde el la aplicación móvil un cambio en el “slider” se usa “servo.to” y se envía con “broadcast” al resto de servidores.

En lo que se refiere a la aplicación móvil en el controlador:

```
.controller('MainCtrl', function($scope, $stateParams, $timeout, $window, socket) {  
    $scope.servo=0;  
    $scope.potentiometer=0;  
    socket.on('init', function (data) {  
        $scope.servo = data.value;  
        //console.log(data);  
    });  
  
    socket.on('from_server', function (data) {  
        //console.log("recieve: "+data.value);  
        $scope.servo = data.value;  
        //TODO Mejorar la forma en la que se actualiza el slider  
        $window.location.reload();  
    });  
  
    socket.on('value', function (data) {  
        console.log("potentiometer: "+data.val);  
        $scope.potentiometer = data.val;  
    });  
  
    $scope.changeRange = function(servos_pass) {  
        //console.log("send: "+servos_pass);  
        $scope.servo = data.value;  
        socket.emit('from_client', { value: servos_pass });  
    };  
});
```

```
}
```

Al cambiar el “slider” se envía la información al programa de Node JS.

Por último en el archivo www/templates/main.html:

```
<div class="range">
    <i class="icon ion-minus-circled"></i>
    <input type="range" ng-model="servo" ng-change="changeRange(servo)" min="0" max="170">
    <i class="icon ion-plus-circled"></i>
</div>
<h3>Actual value {{servo}}</h3>
<h3>{{potentiometer}}</h3>
```

El resultado:

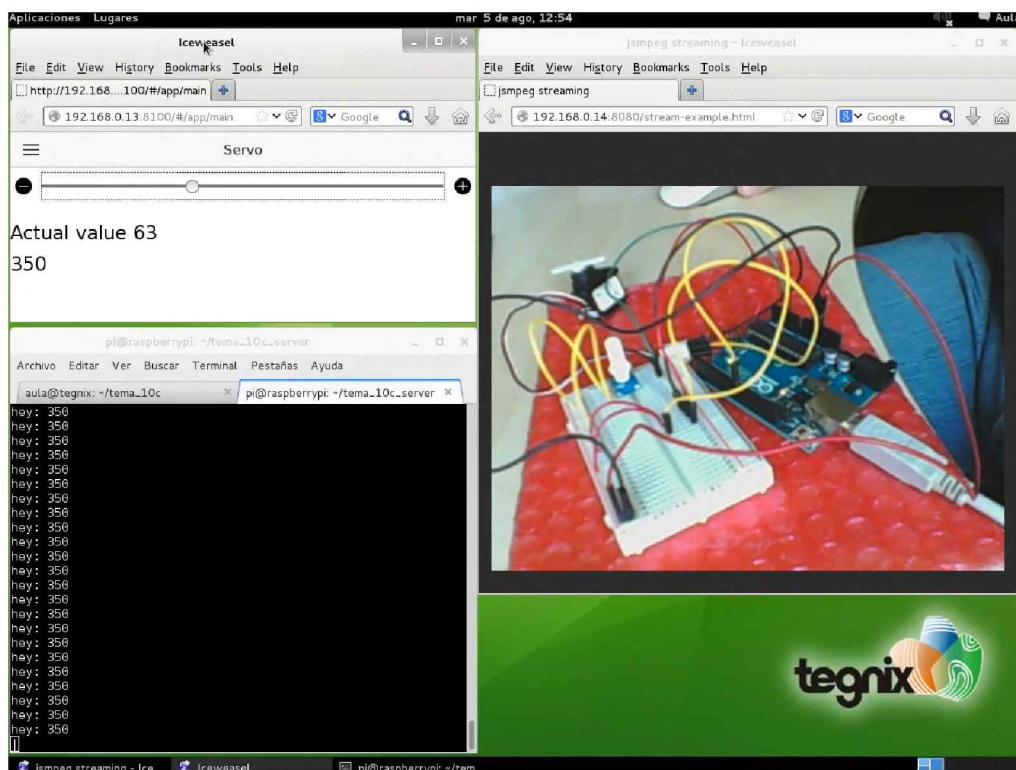


Figura 11.9: Servo ejemplo final

Acerca de los autores

Javier Pardal

Es el gerente de una empresa especializada en Software Libre con más diez años de vida llamada Inforede. En su empresa ofrecen servicios informáticos: redes, administración de sistemas Linux, servidores LAMP, diseño de páginas web, videovigilancia, intranets ... En los últimos tiempos explora con éxito soluciones con Hardware Libre: en especial con Arduino. Por su personalidad y su experiencia dando cursos es un gran divulgador.



Moncho Pena

Actualmente es el CTO de codigo.co.uk una empresa con sede en Londres especializada en desarrollos en base web formada por un grupo interdisciplinar de varias nacionalidades. Durante los últimos años trabajó como Director Técnico para Chaxpert en una multinacional francesa llamada Acticall. Se ha convertido en un especialista en Node JS y en sus interacciones con el Internet de las Cosas. Hace diez años cofundó Inforede, en esa etapa se dedicó a la programación en general. Participó activamente en la Asociación AGASOL y lideró la el proyecto HardProcessing junto a Wireless Galicia. Durante años fue formador en tecnologías donde fue conocido por su paciencia y perseverancia.



Reconocimiento 4.0 Internacional (CC BY 4.0)