Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

Domótica con Arduino y Raspberry

Realizado por
Álvarez Del Barco, José Miguel
Castellano Jiménez, José Antonio

Dirigido por
Portillo Fernández, José Ramón

Departamento
Matemática Aplicada 1

Sevilla - Junio de 2014

Índice general

1 Introducción

En este proyecto nos centraremos en el campo de la **domótica**. El término domótica lo podemos definir como la automatización de un edificio, ya sea hogar u oficina, mediante un sistema que controle características tales como la iluminación, puertas, seguridad y dispositivos de ocio entre otras cosas, con la meta de mejorar nuestro confort y la eficiencia energética del edificio.

Nosotros proponemos un sistema basado principalmente en la comunicación entre Arduino y Raspberry Pi, nuestras dos **piezas claves**. El trabajo conjunto de estos dos dispositivos nos permiten el control de aparatos y objetos más comunes en el hogar y, además, la obtención de datos sobre eventos y elementos climáticos que se desarrollen en el mismo. Por medio de dos aplicaciones, **web y Android**, podremos interactuar con el hogar sin poseer ningún conocimiento del funcionamiento del sistema, haciéndolo accesible a cualquier usuario. A continuación, pasamos a describir brevemente los dispositivos Arduino y Raspberry Pi.

Arduino

El microcontrolador Arduino es uno de los buques insignia del hardware libre de bajo coste diseñado para el ámbito académico y que hoy día es usado tanto por estudiantes como por ingenieros y artistas. Su éxito ha sido tal que podemos ver desde impresoras 3D diseñadas con varios microcontroladores Arduino hasta medidores de pulso cardíaco diseñados con este hardware. Y es que características tales como ser económico, accesible y tener una gran comunidad de usuarios han hecho que seleccionemos a Arduino como el principal componente de nuestro proyecto.

Figura 1.1: Logo de Arduino.

Raspberry Pi

Como ya hemos comentado, Arduino se apoya en Raspberry Pi, un ordenador de placa reducida el cual lo hemos seleccionado por los mismos motivos que Arduino: bajo coste, accesible y posee una gran comunidad de usuarios. Raspberry Pi también compone una parte fundamental en nuestro sistema, ya que mediante éste nos comunicamos con Arduino y, además, proporciona al usuario la aplicación web del sistema domótico.

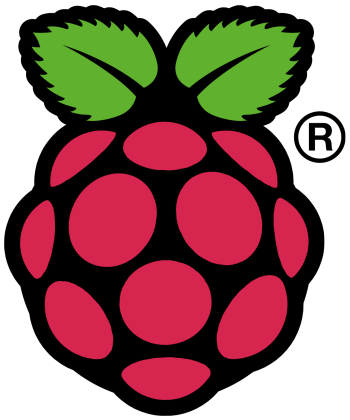


Figura 1.2: Logo de Raspberry Pi.

2 Definición de objetivos

Como se ha comentando anteriormente, proponemos un sistema domótico para el control remoto desde dentro del hogar, así como para recoger datos de lo sucedido en él. Para observar las posibilidades que ofrece nuestro sistema representaremos una casa común por medio de una maqueta de madera.

El objetivo global es, por supuesto, diseñar este sistema que presenta varios objetivos principales y secundarios.

- **Objetivos principales**
  - Controlar elementos de la maqueta. Por ejemplo: iluminación.
  - Obtener datos de eventos y elementos climáticos originados en la maqueta. Por ejemplo, temperatura y humedad.
  - Permitir el control de la maqueta mediante aplicaciones web y Android: Si deseamos que el sistema sea accesible a cualquier persona, necesitamos una interfaz gráfica que facilite al usuario la tarea de controlar la maqueta sin tener conocimientos técnicos previos.
  - Procesar datos recogidos y mostrarlos al usuario en forma de información a través de las aplicaciones web y Android: Los datos por sí solos no sirven de nada, hace falta procesarlos para poderlos entender. Por ejemplo, un sensor de temperatura puede obtener una serie de valores que en sí mismos no indican nada, pero mediante un sencillo cálculo matemático y una visualización gráfica de los mismos podemos interpretar la temperatura durante el día.
- **Objetivos secundarios**
  - Implantar un servidor *streaming*: Aprovechando el potencial de Raspberry Pi, el sistema nos da la oportunidad de reproducir nuestras canciones favoritas desde la aplicación web.

3 Análisis de antecedentes y aportación realizada

La palabra ***domótica*** se acuñó en la década de los 90 en Holanda haciendo referencia a una nueva subdisciplina llamada Tecnología del Hogar que la Universidad de Tecnología de Eindhoven introdujo en el área de investigación de la gerontecnología.

Sin embargo, en los años 70, y debido a la crisis energética, ya se investigaba soluciones para uno de los principales objetivos de la domótica: la eficiencia energética. Fue en 1975 cuando la empresa escocesa Pico Electronics diseñó la primera tecnología de la domótica llamada X10, que constituía un protocolo el cual permite el control remoto de dispositivos domésticos mediante el envío de acciones sencillas como apagar o encender. El protocolo X10 se convirtió en un éxito debido a su gran sencillez y accesibilidad y actualmente es uno de los protocolos más usados, aun así las interferencias por radiofrecuencia o su velocidad de comunicación son sus puntos más débiles.

En los siguientes años aparecieron nuevos protocolos, pero a pesar de la innovación tecnológica y la creciente industria sobre la domótica, aún se sigue considerando que está enfocada a personas de clase alta. Además, la carencia de un protocolo único y simplificado, la necesidad de instalaciones personalizadas, más el alto coste que supone el establecimiento de un sistema domótico provocan la caída del número de consumidores interesados en los servicios que puede brindarles la domótica.

A lo referente con nuestro proyecto, gracias a los conocimientos anteriormente obtenidos del funcionamiento del microcontrolador Arduino y a la mejora sustancial de la técnica de programación obtenida tanto en el ámbito académico como en el ámbito laboral, vamos a realizar un sistema domótico cuyo núcleo principal esté formado por **Arduino y Raspberry Pi**. Es decir, un sistema cuyos componentes sean **económicos** y, por tanto, esté al alcance de una **mayor cuota de mercado**, un sistema que entra perfectamente en la dinámica de **DIY** ("Do It Yourself"), lo cual le da un punto a su favor. Para ello, investigaremos la comunicación entre ambos dispositivos y ofreceremos una interfaz gráfica basada en dos aplicaciones, **una web y otra Android**, que acerca dicho sistema a cualquier usuario sin conocimientos técnicos de ningún tipo. Estas aplicaciones se comunicarán con Raspberry Pi mediante los famosos protocolos **TCP/IP**.

Por último, debemos añadir que aunque el sistema esté enfocado al control de una maqueta es totalmente factible su funcionamiento en un escenario real, por supuesto con previas modificaciones para adaptarlo a los requisitos de energía que exigen los dispositivos domésticos.

4 Análisis temporal y de costes de desarrollo

Hemos de tener en cuenta que en un proyecto de investigación como es el caso, es complicado realizar un correcto análisis temporal y de costes de desarrollo. Es por ello que hemos adaptado esta tarea para que se ajuste a nuestro modo de trabajo.

Para nuestro proyecto, hemos adoptado un ciclo de desarrollo iterativo, subdividiendo la funcionalidad total deseada en ciclos o iteraciones. Al final de cada ciclo o iteración debemos contar con componentes totalmente integrados y funcionales. Siguiendo esta metodología de trabajo, hemos realizado la planificación temporal al principio de cada iteración, dividiendo la tarea en correspondencia con las principales etapas de un desarrollo software (análisis, diseño, implementación y pruebas). Por lo tanto, para realizar una estimación temporal hemos planificado por separado cada tarea y le hemos aplicado un porcentaje que representa la parte correspondiente a cada etapa del desarrollo.

Hemos estimado que la implementación de todos los componentes del sistema final requiere **300 horas**, a las cuales aplicamos los siguientes porcentajes:

- **Análisis**: 10 %.
- **Diseño**: 15 %.
- **Implementación**: 65 %.
- **Pruebas**: 10 %.

Obteniendo así los siguientes resultados aproximados:

- Un total de **30 horas** dedicadas al **Análisis**.
- Un total de **45 horas** dedicadas al **Diseño**.
- Un total de **195 horas** dedicadas a **Implementación**.
- Un total de **30 horas** dedicadas a **Pruebas**.

Por otra parte, hemos estimado que el coste en horas invertido en redactar la **documentación** sería de **200 horas**, a las que añadimos **20 horas de configuración de dispositivos**, otras **25 horas** de trabajo sobre la **maqueta**, y **16 horas** que estimamos emplear en **configurar los entornos de desarrollo** de las distintas tecnologías.
También debemos tener en cuenta el tiempo dedicado a la gestión del proyecto, que se divide en reuniones con el tutor, coordinación y comunicación de los autores del proyecto, seguimiento del proyecto desde la plataforma ProjETSII y compra de materiales. Hemos estimado en **100 horas** este apartado.En total hacen una suma de **661 horas**.

Una vez finalizado el proyecto, hemos realizado un recuento de las tareas y el tiempo empleado asciende hasta las **728 horas**. El desvío en la planificación se debe, como

mencionamos al principio de este capítulo, a la dificultad de estimar la duración de un proyecto de estas características y a la inexperiencia de los componentes del proyecto. Existe una diferencia de carga de trabajo entre los miembros debido a que uno de los cuales realizaba prácticas en empresa a la vez de este proyecto, sin embargo, los días de trabajo han sido los mismos para ambos componentes. Podemos ver el desglose de las tareas en la carpeta **Tareas** del DVD adjunto.

4.1 Análisis de costes

Como ya hemos mencionado anteriormente, es complicado elaborar una buena estimación **a priori** de costes en un proyecto de investigación. Por tanto, nuestra estrategia ha sido fijarnos un tope presupuestal que, por acuerdo, sea igual al doble de los componentes básicos necesarios para proyecto (placa Arduino y Raspberry Pi).

4.1.1 Recursos no humanos

Procedemos a incluir la siguiente tabla de costes:

Descripción	Proveedor	Unidades	Fecha realización pedido	Fecha de recepción de envío	Coste unidad	Total
Arduino Starter Kit	Cetronic S.L.	1	18/06/2013	21/06/2013	96,05 €	107,64 €
Maqueta						
Base maqueta	Leroy Merlin	1	29/01/2014	29/01/2014	9,23 €	9,23 €
Pack Bisagras latonadas 25x25 mm	Leroy Merlin	2	29/01/2014	29/01/2014	3,55 €	7,10 €
Pack secciones maqueta	Carpintería	1	5/02/2014	14/02/2014	20,00 €	20,00 €
Componentes eléctricos y electrónicos						
Arduino Mega 2560	EasyElectronica	1	19/01/2014	23/01/2014	18,00 €	18,00 €
RPI modelo B rev 2., SD 8GB, cargador, carcasa	CJ-Elektronik	1	30/01/2014	10/02/2014	53,99 €	59,99 €
Breadboard, cable 3m	Multiluz	1	30/01/2014	30/01/2014	4,00 €	4,00 €
Cable Jack a USB para Arduino	Electronicca	1	3/02/2014	17/02/2014	2,90 €	3,70 €
Útiles de electrónica (leds, resistencias, alicates...)	MLEON	1	4/02/2014	4/02/2014	12,00 €	12,00 €
Ventiladores	-	3	4/02/2014	4/02/2014	1,00 €	3,00 €
40 cables jumper	Ofertas*com	1	4/02/2014	10/02/2014	3,75 €	5,00 €
Edimax EW-7811UN Wireless nano USB Adapter	Envizage Solutions Limited	1	6/02/2014	21/02/2014	9,23 €	11,53 €
Keypad Arduino 12 teclas	Elictronica	1	6/02/2014	11/02/2014	2,60 €	3,60 €
40 cables jumper	Ofertas*com	1	7/02/2014	11/02/2014	3,90 €	5,95 €
L298N Dual H Bridge		2	7/02/2014	7/02/2014	6,00 €	12,00 €
100 cables 6cm	Ebay	1	14/02/2014	24/02/2014	1,99 €	2,37 €
Arduino Mega 2560	Ebay	1	20/02/2014	24/02/2014	20,00 €	20,50 €
L298N Dual H Bridge Motor Drive Controller Board Module	Ebay	1	5/03/2014	11/03/2014	5,70 €	5,70 €
40 cables jumper	Ebay	1	12/03/2014	19/03/2014	3,00 €	3,00 €
Sensor humedad y temperatura DHT11	Ebay	1	24/03/2014	1/04/2014	1,85 €	2,35 €
Sensor de Ultrasonido HC-SR04	Ebay	1	24/03/2014	4/04/2014	1,76 €	2,76 €
Sensor MQ-2	Ebay	1	29/03/2014	4/04/2014	5,60 €	5,60 €
Herramientas						
Soldador	Ferretería Reina Mercedes	1	7/02/2014	7/02/2014	9,40 €	9,40 €
Estaño	Ferretería Reina Mercedes	1	7/02/2014	7/02/2014	2,00 €	2,00 €
Regleta	Ferretería Reina Mercedes	1	7/02/2014	7/02/2014	5,50 €	5,50 €
Adhesivo Patex, 20 ángulos de ensamblaje, 2 pack de tornillos, 1 caja clasificadora	Leroy Merlin	1	17/02/2014	17/02/2014	15,50 €	15,50 €
2 visagras, cinta aislante, tornillos	Ferretería Reina Mercedes	1	13/03/2014	13/03/2014	2,00 €	2,00 €
Juego de brocas	Ferretería Reina Mercedes	1	21/03/2014	21/03/2014	1,80 €	1,80 €
Multímetro y pila 9V	Ferretería Reina Mercedes	1	21/03/2014	21/03/2014	7,00 €	7,00 €
	Coste estimado del Proyecto	335,26 €				
	Coste total del Proyecto	368,22 €				

Figura 4.1: Tabla de costes materiales

Para el correspondiente análisis de costes no hemos tenido en cuenta los recursos ya disponibles (portátiles, dispositivos móviles, etc.). Por tanto, el coste de la inversión ha

sido de **368,22 €** suponiendo una diferencia de **32.96 €** respecto a la estimación inicial, aproximadamente el **9.8 %**.

4.1.2 Recursos humanos

Para calcular el coste de los recursos humanos hemos asumido que los integrantes del proyecto se ajustan al perfil de programador, el cual tiene un coste estimado de 25 €/hora.

Perfil	Horas	€/Hora	Total
Programador	661	25	16525,00 €

Cuadro 4.1: Costes estimados asociados a recursos humanos.

Perfil	Horas	€/Hora	Total
Programador	728	25	18220,50 €

Cuadro 4.2: Costes reales asociados a recursos humanos.

La diferencia de los datos aproximados con respecto a los datos reales tiene como origen la inexperiencia de los miembros del proyecto a la hora de planificar y los distintos obstáculos que nos hemos ido encontrando a la hora de desarrollar el proyecto.

5 Análisis de requisitos. Herramientas utilizadas y justificación

5.1 Análisis de requisitos

A continuación, especificamos los requisitos que debe cumplir el sistema:

5.1.1 Requisitos funcionales

- El sistema debe permitir el control remoto de una maqueta que representa una vivienda.
- El sistema debe proporcionar una interfaz gráfica para el control de la maqueta y para la comunicación con el usuario.
- El sistema debe proveer de un servicio de login para el acceso al control de la maqueta.
- El sistema debe permitir el control de la iluminación de la maqueta.
- El sistema debe permitir la apertura y cierre de puertas.
- El sistema debe proporcionar un panel físico para la seguridad de la puerta principal.
- El sistema debe permitir el manejo de electrodomésticos.
- El sistema debe permitir el control de la calefacción y aire acondicionado de la maqueta.
- El sistema debe recoger distintos datos del entorno, como temperatura, humedad, etc., para su posterior procesamiento.
- El sistema debe notificar al usuario de los eventos más importantes.
- El sistema debe permitir al usuario almacenar archivos de música
- El sistema debe permitir al usuario reproducir los archivos de música almacenados.

5.1.2 Requisitos no funcionales

- **Requisitos del producto**
  - **Requisitos de usabilidad**
    - La curva de aprendizaje para el manejo de las interfaces de control ha de ser baja.
    - El usuario debe recibir información acerca de las acciones que realiza.

- El usuario debe poder comunicarse con el sistema mediante una aplicación web y una aplicación Android.
- El usuario debe poder realizar una acción de manejo sin impedimentos.
- La interfaz gráfica debe ser amena para el usuario, intentado disminuir en lo posible la carga cognitiva.

Requisitos de eficiencia

- El tiempo de respuesta del sistema no debe ser mayor de 2 segundos.
- Los componentes eléctricos y electrónicos que son controlados por Arduino deben presentar valores adecuados de intensidad y voltaje.

Requisitos organizacionales

Requisitos de entrega

- La fecha de finalización del proyecto no debe superar el 3 de Junio de 2014.
- La memoria del proyecto debe entregarse como un archivo único en formato PDF el día 3 de Junio de 2014.
- La presentación del proyecto debe entregarse como un archivo único el día 16 de Junio de 2014.

Requisitos de implementación

- El sistema debe basarse en un modelo Maestro/Esclavo.
- La comunicación entre los elementos principales, que intervienen en el sistema, debe ser bidireccional.
- La comunicación entre Raspberry Pi y Arduino debe ser mediante enlace físico USB (Tipo A - Tipo B).
- Raspberry Pi debe ser quien inicie la comunicación con Arduino, salvo en casos excepcionales.
- La comunicación entre Raspberry Pi y Arduino se realizará en lenguaje PHP 5 o superior, y Python 2.7.
- El usuario debe comunicarse con Raspberry Pi de manera remota mediante tecnología WI-FI.
- Raspberry Pi debe disponer de un repositorio para poder reproducir música.
- Raspberry Pi debe disponer de una base de datos MySQL 5.0 o superior, para registrar datos de Arduino y almacenar las credenciales del usuario.
- El sistema debe implementar un servidor web Apache 2.0 o superior, en el dispositivo Raspberry Pi.
- El servidor web debe apoyarse en el lenguaje PHP 5 y en la técnica Ajax para brindar páginas web dinámicas.
- Las páginas web deben desarrollarse en HTML5, CSS3 y JavaScript.
- El sistema puede hacer uso de librerías de terceros.
- En la placa Arduino debe encontrarse el código que procesa y ejecuta los comandos de Raspberry Pi.
- El código de la placa de Arduino debe estar escrito en C/C++.
- La aplicación de Android debe estar enfocada a la plataforma Android 4.0 o superior.

- El lenguaje usado para la aplicación Android debe ser nativo.

- **Requisitos externos**

  - **Requisitos éticos**

    - El sistema debe estar compuesto por software libre.
    - El uso de software de terceros debe especificarse explícitamente en la memoria del proyecto.

  - **Requisitos legislativos**

    - Las credenciales del usuario final para el control del sistema no deben estar expuestas al público.

5.2 Herramientas usadas y justificación

A continuación, definiremos brevemente las herramientas más importantes utilizadas en el proyecto con las principales funciones de cada una en el mismo y su correspondiente justificación.

5.2.1 Hardware

En el orden alfabético, los dispositivos hardware más destacados son:

Adaptador USB WI-FI Edimax EW-7811UN

Con el adaptador WI-FI Edimax podemos conectarnos remotamente a Raspberry Pi. El modelo EW-7811UN es totalmente **compatible** con el sistema operativo Raspbian por lo que no requiere instalación manual ni configuración previa, simplemente conectar a un puerto USB y listo. Además, este modelo es compacto, económico y es compatible con el estándar inalámbrico 802.11b/g/n.

Figura 5.1: Adaptador USB WI-FI Edimax EW-7811UN.

Arduino

Arduino es un microcontrolador de hardware y software libres, de bajo coste que nos permite llevar a cabo el **control** de los componentes intregrados en la maqueta. Con Arduino podemos obtener información del entorno así como controlar periféricos mediante distintos tipos de pines que posee. Los tipos de pines más importantes para este proyecto son los siguientes:

- **Pines digitales**: Estos pines podemos configurarlos como entradas o salidas mediante la función de *pinMode()*. Los ledes[1] o motores paso a paso, entre otros elementos, pueden ser controlados mediante estos pines.
- **Pines analógicos**: Pines destinados a la lectura de sensores analógicos, aunque también pueden ser usados como pines digitales. La lectura de estos pines se realiza con la función de Arduino *analogRead()*, devolviendo valores entre 0 y 1023. Por medio de estos pines podemos obtener datos de sensores de temperatura, iluminación, etc.
- **Pines PWM**: Pines digitales dedicados a simular una salida analógica con una salida digital. Mediante este tipo de pines podemos aumentar o disminuir la iluminosidad de ledes.

Existe una gran variedad de modelos de Arduino. Para nuestro sistema domótico usamos el modelo **Mega 2560 rev. 3** debido a que posee un gran número de pines: 54 pines digitales y 16 pines analógicos. Esta cantidad de pines satisface la necesidad de conectar múltiples componentes a Arduino que con otras versiones no podríamos realizar. Sin embargo, utilizamos además Arduino UNO para que ambos integrantes del grupo podamos trabajar paralelamente.

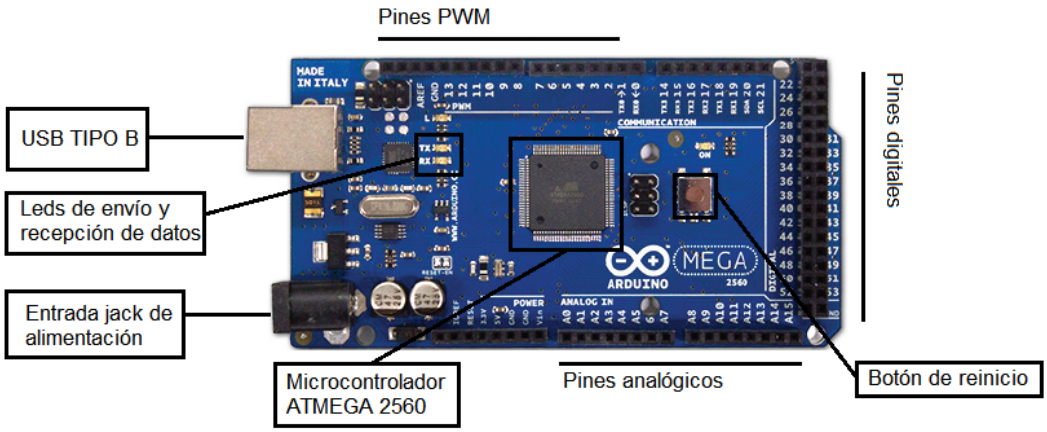


Figura 5.2: Arduino Mega 2560 rev. 3.

Pantalla LCD

Las pantallas son uno de los principales medios para mostrar información al usuario. En nuestro caso hemos recurrido a la pantalla LCD WINSTAR WH1602B-TMI-ET incluida en el Starter Kit de Arduino para informar al usuario de la interacción que realiza con el teclado numérico de la entrada principal. Esta pantalla es alfanumérica y puede mostrar dos filas de 16 caracteres cada una. También es compatible con el controlador Hitachi HD44780 lo que nos permite controlarla fácilmente con el entorno de desarrollo de Arduino.

Figura 5.3: Pantalla LCD WINSTAR WH1602B-TMI-ET.

[1]Plural de led según la RAE.

Raspberry Pi

Raspberry Pi es un minicomputador de bajo coste diseñado para el ámbito académico con el objetivo de cubrir las funciones que un ordenador personal puede dar a un precio muy competitivo. Para poder hacer uso del mismo, debemos instalar uno de los sistemas operativos libres compatibles existentes.

En cuanto al proyecto, este dispositivo nos ayuda a enviar órdenes a Arduino de manera inalámbrica (WI-FI) y ofrece la aplicación web al usuario por medio del servidor web instalado en él. Aunque Arduino disponga de los módulos de servidor web y WI-FI es más **económico** adquirir Raspberry Pi y, además, este último **agiliza** la tarea de implementar todo lo referente al diseño web. Como punto negativo, la inclusión de Raspberry Pi al proyecto nos lleva a investigar la comunicación entre dicho dispositivo y Arduino.

Por último, hemos optado por elegir el **modelo B** de Raspberry Pi debido a que incluye dos puertos USB y un puerto Ethernet. Uno de los puertos USB está dedicado a la comunicación con Arduino; el otro lo utilizamos para conectar el adaptador WI-FI. El puerto Ethernet lo usábamos mientras no disponíamos de adaptador WI-FI.

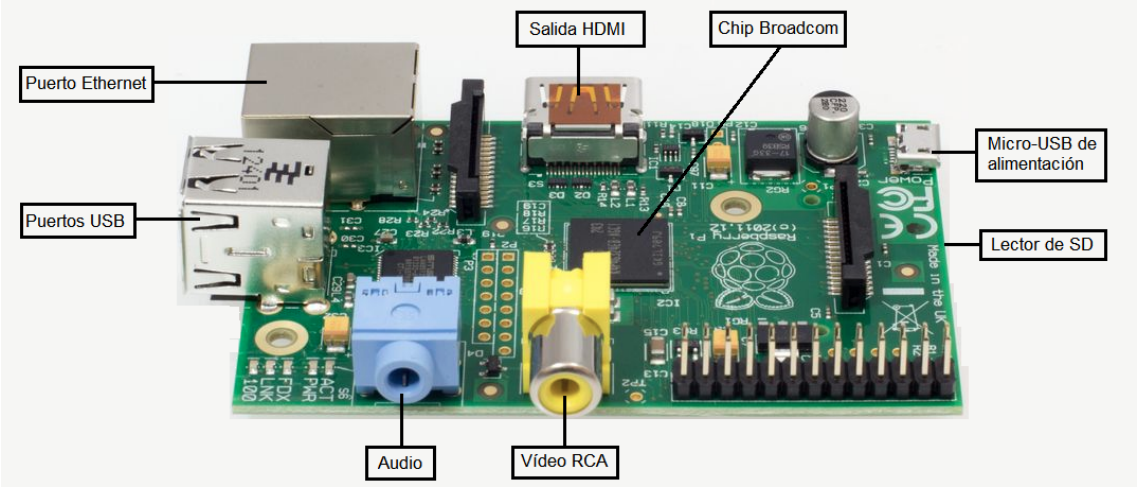


Figura 5.4: Raspberry Pi modelo B rev. 2.0.

5.2.2 Componentes eléctricos y electrónicos

En orden alfabético, los componentes principales son los siguientes:

Controlador L298N Dual H Bridge

El controlador L298N es un chip con el cual mantenemos la energía, que controla un motor, **separada** de Arduino. Como Arduino no puede generar suficiente energía como para conectarlo directamente a motores, nos valemos del controlador para hacer funcionar motores DC o motores paso a paso mediante Arduino. Además, el uso de este controlador suaviza el movimiento del motor que vayamos a utilizar.

Entre la gran variedad de controladores, el L298N es uno de los más **económicos** permitiendo controlar un motor paso a paso o dos motores DC al mismo tiempo.

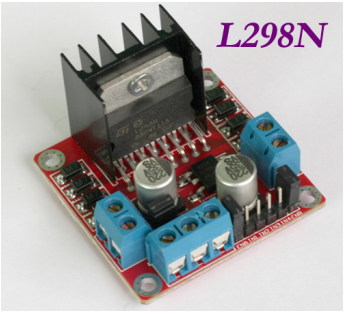


Figura 5.5: Controlador L298N Dual H Bridge.

Led

Por su sencilla integración con Arduino mediante pines digitales y su precio asequible, el led ha sido una buena opción para simular la iluminación de un hogar. Como existe una gran variedad, hemos elegido ledes de tipo miniatura de 5 mm (ultra-high-output) y 3 mm (standard), con voltajes de 3,4 V y 2,1 V, respectivamente.

Figura 5.6: Led ultra-high-output de 5 mm.

Micrófono de contacto (Piezo speaker)

El micrófono de contacto permite crear y detectar sonidos mediante **piezoelectricidad**, un efecto donde determinados cristales cambian de forma cuando le aplicamos electricidad. Aplicando una señal eléctrica a una frecuencia correcta, los cristales pueden emitir sonido. Para poder controlar el micrófono debemos usar los **pines PWM**, para un mejor sonido, y la función *tone()* de Arduino.

El micrófono que hemos usado para este proyecto es bastante barato, pero cumple con las tareas que exigimos, como la función de alarma. Se trata del modelo PKM17EPP-4001-B0.

Figura 5.7: Micrófono de contacto serie PKM17EPP.

Motor paso a paso bipolar

"El motor paso a paso es un dispositivo electromecánico que convierte una serie de impulsos eléctricos en desplazamientos angulares discretos, lo que significa que es capaz de avanzar una serie de grados (paso) dependiendo de sus entradas de control. Dicho motor se comporta de la misma manera que un conversor digital-analógico (D/A) y puede ser gobernado por impulsos procedentes de sistemas lógicos". Definición obtenida de *Wikipedia*.

La ventaja de usar este tipo de motores es que presenta una **alta precisión** en sus movimientos frente a otros como los motores de corriente continua, es por ello que lo utilizamos para la apertura de la puerta del garaje.

Figura 5.8: Motor paso a paso bipolar.

Sensor DHT11

El sensor DTH11 es uno de los sensores de humedad más populares y económicos en el mercado, el cual permite la lectura de la humedad relativa comprendida entre 20 % y 90 %, así como la lectura de temperatura en un rango de 0 °C a 50 °C. Ambas lecturas devuelven números enteros y se realizan fácilmente desde un pin **digital** o **analógico** mediante el uso de la librería diseñada por **Adafruit** para este sensor.

Tras un mal funcionamiento del sensor TMP-36, comenzamos a utilizar el sensor DHT11 para recoger datos de la temperatura.

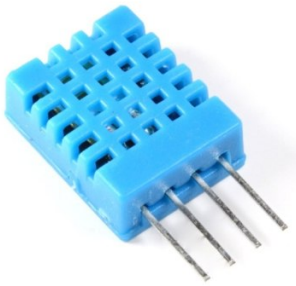


Figura 5.9: Sensor DHT11.

Sensor HC-SR04

El sensor HC-SR04 de **bajo coste** puede detectar la presencia de objetos devolviendo la distancia a la que se encuentra de los mismos. Puede trabajar en un rango de 2 cm a 400 cm con resultados satisfactorios, aunque puede detectar objetos a mayor distancia pero el fabricante no asegura un buen rendimiento.

El código necesario para interpretar los datos que recoge lo podemos realizar mediante la librería diseñada por **J. Rodrigo**, con la que se obtiene mismos resultados que si lo hacemos mediante cálculos matemáticos traduciendo el tiempo transcurrido desde que envía la onda hasta que la recibe teniendo en cuenta la velocidad del sonido a temperatura ambiente.

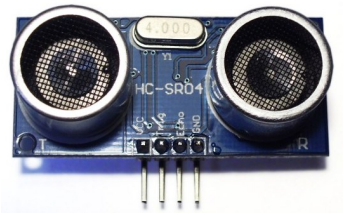


Figura 5.10: Sensor HC-SR04.

Sensor MQ-2

El sensor MQ-2 es uno de los sensores de gases combustibles **más baratos**. Tiene la capacidad de detectar concentraciones de diferentes gases entre 300 y 10000 ppm (partes por millón) con una sensibilidad muy aceptable. Los gases que puede detectar son los siguientes:

- Dihidrógeno (H_2).
- Gases licuados del petróleo (GLP).
- Metano (CH_4).
- Monóxido de carbono (CO).
- Alcohol

Quizás la parte negativa de este sensor es que no podemos distinguir a qué tipo de gas pertenece la cantidad detectada.

Si utilizamos el sensor acoplado a su correspondiente módulo podremos agilizar el proceso de conexión con Arduino. En nuestro proyecto utilizamos MQ-2 para detectar el **gas butano** perteneciente a los GLP. A partir de una determinada cantidad de butano (2000 ppm), activaremos la alarma.

Figura 5.11: Módulo y sensor MQ-2.

Sensor TMP-36

TMP-36 es uno de los sensores analógicos más populares y sencillos de usar. Conectándolo a un pin analógico de Arduino obtenemos la temperatura en un rango de -40 °C y 100 °C. Además de su rápida implementación y sencillo uso, es muy barato.

Sin motivo aparente el sensor TMP-36 **dejó de funcionar**, frecuentemente obtenía valores incoherentes como 55 °C de temperatura, por lo que decidimos medir la temperatura con el sensor DHT11.

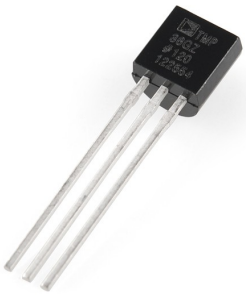


Figura 5.12: Sensor TMP-36.

Servomotor

El servomotor es un motor de corriente continua que permite controlar su posición mediante un eje. Dicha posición oscila en un rango establecido entre 0° y 180°, aunque también existen servomotores con un rango distinto, incluso de rotación continua. Estos motores son muy baratos, poseen una **gran fuerza y precisión** permitiéndonos implantar funcionalidades como la apertura y cierre de la puerta principal.

Figura 5.13: Servomotor de 5 V.

Teclado numérico

Para añadir la posibilidad de apertura de la puerta principal mediante un panel físico, hemos pensado en utilizar un teclado numérico de membrana de 12 teclas muy popular y barato en el mundo de Arduino.

Figura 5.14: Teclado 3x4.

Transistor MOSFET IRF520

El transistor IRF520 nos sirve como **interruptor** para controlar el estado de componentes que exigen de una alimentación externa, por ejemplo, un ventilador. Este transistor es **muy recomendado** en proyectos con Arduino no sólo por su precio, sino además, por la conmutación que la realiza a una frecuencia muy similar al procesador de Arduino.

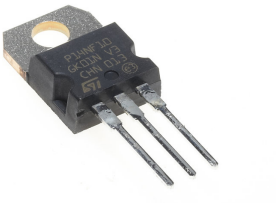


Figura 5.15: Transistor MOSFET IRF520.

Ventilador Brushless CC

Los ventiladores Brushless CC son aquellos ventiladores que incorporan un motor de corriente Brushless, el cual no dispone de escobillas adquiriendo mayor eficiencia y una menor probabilidad de fallo. Con este ventilador pretendemos representar la ventilación de un hogar cualquiera.

Figura 5.16: Ventilador Brushless CC.

5.2.3 Software

Recursos software más importantes ordenados alfabéticamente:

Ajax

Ajax es un conjunto de técnicas orientadas al diseño web cuyo objetivo es realizar aplicaciones web que puedan comunicarse con el servidor sin necesidad de interferir en el funcionamiento y la apreciación de la página web, es decir, comunicarse de manera **asíncrona**.

Su implementación en nuestro proyecto ha sido un proceso cómodo con el uso de la librería jQuery. Con Ajax, el servidor Apache de Raspberry Pi **recibe las órdenes** originadas por la interacción entre usuario y aplicación web. Entre otras funciones, Ajax también actualiza información mostrada en dicha aplicación de una manera agradable de cara al usuario.

Android

Android es un sistema operativo de código abierto diseñado por Android Inc., enfocado a dispositivos móviles. Su popularidad y gran comunidad de desarrolladores han sido suficientes motivos para incluir esta herramienta en nuestro proyecto.

Por medio de Android, hemos diseñado una aplicación para que el usuario pueda interactuar con la maqueta de una forma amena y rápida desde dispositivos tales como *smartphones* o *tablets* basados en Android. Debido al amplio número de versiones de Android, hemos decidido que nuestra aplicación requiera **Android 4.0** o superior, pues el número de dispositivos con versiones inferiores es aproximadamente del 20 % (ver imagen[2] adjunta procedente de la web oficial de desarrolladores Android) y está disminuyendo progresivamente.

[2]Datos recogidos el día 1 de abril de 2014.

Version	Codename	API	Distribution
2.2	Froyo	8	1.1%
2.3.3 - 2.3.7	Gingerbread	10	17.8%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	14.3%
4.1.x	Jelly Bean	16	34.4%
4.2.x		17	18.1%
4.3		18	8.9%
4.4	KitKat	19	5.3%

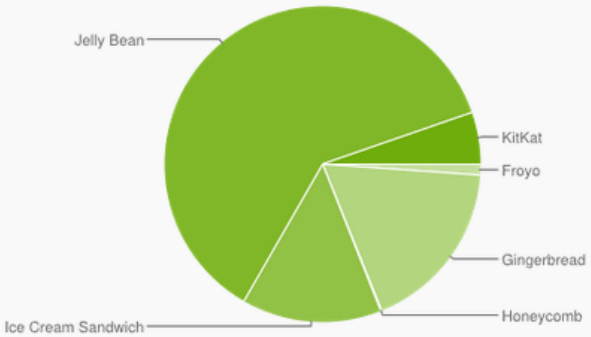


Figura 5.17: Porcentajes de dispositivos Android según la versión del sistema.

Figura 5.18: Logo de Android.

Android Studio

Android Studio es un entorno de desarrollo integrado gratuito desarrollado por Google Inc.. Este software nos **facilita** enormemente la tarea de instalar todo lo necesario para comenzar a programar en Android.

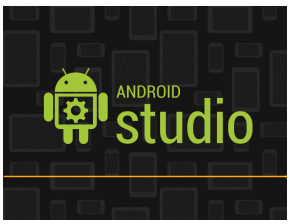


Figura 5.19: Logo de Android Studio.

Apache

Apache es un servidor web HTTP libre diseñado por Apache Software Foundation. Alojado en Raspbery Pi, Apache nos proporciona la aplicación web mediante la cual el usuario puede interactuar con la maqueta. Ya que Raspbian soporta Apache y requiere de una

instalación y puesta en marcha muy simples, ha sido fácil optar por esta herramienta.

Figura 5.20: Logo de Apache.

Arduino IDE

Para poder realizar programas compatibles con Arduino y cargarlos en el dispositivo hemos usado el entorno de desarrollo oficial en su versión 1.0.5, descrito en el apéndice *Entorno de desarrollo de Arduino*. El lenguaje de programación para este entorno es conocido como *Arduino*, al igual que el dispositivo, basado en **C/C++**.

Las principales librerías de programación ya vienen integradas con el entorno. Si se necesita una librería de terceros, simplemente la añadimos en el directorio `/libraries`. Para este proyecto nos hemos apoyado en las siguientes librerías:

- ***DHT.h***: Librería externa desarrollada por la empresa **Adafruit Industries**, cuyo uso es gratuito y nos ha permitido controlar el sensor de temperatura y humedad DHT11.
- ***Keyboad.h***: Librería gratuita externa desarrollada por la comunidad de Arduino que nos permite el control del teclado numérico.
- ***LiquidCrystal.h***: Librería incluida en el entorno que nos ayuda a mostrar información en la pantalla LCD.
- ***Servo.h***: Librería incluida en el entorno que nos permite controlar servomotores.
- ***Stepper.h***: Librería incluida en el entorno que nos permite el control de motores paso a paso.
- ***string.h***: Librería incluida en el entorno utilizada para dividir los comandos que recibe Arduino en los distintos campos que los forman.
- ***Ultrasonic.h***: Librería externa gratuita desarrollada por **J. Rodrigo**, cuya finalidad es controlar el sensor de ultrasonidos HC-SR04. El sensor fue implementando también con nuestro propio código y al comprobar que daba resultados similares decidimos optar por la librería, pues el código se reducía a una sola línea.

CSS

CSS es un lenguaje para definir la apariencia de documentos desarrollados en lenguajes de marcas, como por ejemplo HTML. Al tener conocimientos previos de este lenguaje, nos hemos decantado por él para separar el estilo del propio contenido de la aplicación web.

Figura 5.21: Logo de CSS3.

FileZilla

FileZilla es un cliente y servidor FTP de licencia GNU. Con el cliente de FileZilla podemos subir archivos al servidor FTP de Raspberry Pi y de esta manera poder programar en nuestros ordenadores personales cómodamente sin necesidad de hacerlo sobre Raspberry Pi.

Figura 5.22: Logo de FileZilla.

Fritzing

Fritzing es un software gratuito dedicado al diseño electrónico permitiendo documentar nuestros prototipos y compartirlos con otros usuarios. Una de las razones para usarlo, ademas de ser software libre, es que se trata del programa más completo para diseñar circuitos con **Arduino**.

Figura 5.23: Logo de Fritzing.

HTML

HTML es un lenguaje de marcas para la creación de documentos, páginas web y programas informáticos. En nuestro proyecto hemos usado HTML en su versión 5 para la

creación y agrupación de los elementos de la aplicación web.

Figura 5.24: Logo de HTML5.

JavaScript

JavaScript es un lenguaje de programación interpretado y con un mínimo control de tipo de variables. Debido a su popularidad, **facilidad de uso** y multitud de librerías se ha convertido en una herramienta esencial en el diseño de la aplicación web para nuestro proyecto.

Con JavaScript hemos podido añadir funcionalidades muy variadas a nuestra aplicación web. A continuación, detallamos las herramientas que hemos utilizado basadas en JavaScript:

- **elFinder**: *Widget* implementado en jQuery por Studio-42 con el que podemos subir nuestros archivos de música para reproducirlos con el reproductor de la aplicación web.
- **iPicture**: *Plugin* gratuito basado en jQuery y desarrollado por Sara D'Alia. Nos ha permitido colocar iconos sobre el plano de la casa, los cuales nos informan del nombre de la habitación en la que estamos situados de una forma elegante.
- **JPlaylister**: *Widget* gratuito basado en jQuery y PHP que nos ha permitido incluir un reproductor de música en la aplicación web. El *widget* ha sido desarrollado por Chapman IT.
- **jQuery**: Librería gratuita desarrollada originalmente por John Resig que nos permite navegar por la estructura de un documento HTML, crear animaciones o controlar eventos, entre otras cosas, de una manera muy rápida y eficaz. El principal uso que le hemos dado a esta librería consiste en la gestión de eventos con la interacción entre usuario y página web.
- **jQuery UI**: Librería gratuita basada en jQuery desarrollada por jQuery UI Team. Entre sus funciones, elegimos los botones desplazamiento con los que podemos encender ledes de manera gradual.
- **jQWidgets**: Librería gratuita basada en jQuery y HTML5. Nos ayuda a crear botones con estado (on/off), menús horizontales y gráficos de datos.
- **noty**: *Plugin* gratuito basado en jQuery desarrollado por Nedim Arabaci. Con este *plugin* notificamos al usuario de errores de conexión a través de la aplicación web.

Figura 5.25: Logo de JavaScript.

LaTeX

LaTeX es un sistema de elaboración de documentos para tipografía de alta calidad. Fue desarrollado originalmente por Leslie Lamport mediante instrucciones de TeX y es el estándar de facto para la publicación de documentos científicos.

Decidimos usar LaTeX para redactar la memoria del proyecto debido a que presenta una serie de ventajas bastante atractivas a pesar de que conlleve un tiempo de **aprendizaje** para su dominio:

- Se trata de **software libre**.
- Proporciona un **aspecto profesional** a los documentos.
- **Automatiza acciones** haciendo que la tarea de redactar un documento sea más ágil y sencilla.
- Permite crear fácilmente **documentos estructurados**.
- Da la posibilidad de **crear comandos** que realicen acciones más complejas.

Figura 5.26: Logo de LaTeX.

Mercurial

Mercurial es un sistema de control de versiones de uso público desarrollado por Matt Mackall. Nos permite versionar nuestro código como cualquier otro sistema con el mismo objetivo pero añade algunas características como poder hacer *commits* sin tener acceso al **servidor principal** o volver a una versión anterior de cualquier archivo sin la necesidad de estar conectado al servidor principal.

Figura 5.27: Logo de Mercurial.

Microsoft Project

Microsoft Project es un software de **gestión de proyectos** de gran envergadura ampliamente enfocado al ámbito laboral desarrollado por Microsoft.

A diferencia de ProjETSII y acorde con nuestras necesidades, Microsoft Project nos da la posibilidad de incluir tareas realizadas por varias personas (recursos), además de aportar mayor información del estado del proyecto en cada momento mediante diagramas y gráficos.

MySQL

MySQL es un sistema de gestión de base de datos gratuito diseñado originalmente por MySQL AB. El principal motivo que ha conllevado a la elección de esta herramienta es la compatibilidad con el sistema operativo Raspbian. Entre las distintas versiones, hemos optado por **MySQL 5.4.4**, puesto que a partir de la versión 5.0, las actualizaciones han ido incluyendo notorias **mejoras** en cuanto al rendimiento con el uso de **PHP 5**.

Al tratarse de un sistema de bases de datos, las funciones de esta herramienta tienen que ver con el manejo de datos:

- **Almacenamiento de notificaciones**: Guarda las notificaciones que Arduino envía a Raspberry Pi como confirmación de la recepción de las órdenes.
- **Credenciales**: Almacena el nombre y contraseña de usuario que son requeridos a la hora de conectarse a la interfaz gráfica de usuario, ya sea web o Android.
- **Datos de sensores**: Guarda los datos de los sensores conectados a Arduino para devolverlos, posteriormente, cuando sea necesario.
- **Registros innecesarios**: Elimina datos almacenados que ya no son necesarios.

Figura 5.28: Logo de MySQL.

PHP

PHP es un lenguaje de programación de **código abierto** diseñado por Rasmus Lerdorf para el desarrollo web, aunque también se usa para otros propósitos como aclararemos después. Entre sus puntos fuertes destacamos:

- Lenguaje fácil de aprender.
- Seguridad: El código PHP es interpretado por el servidor, siendo invisible para el cliente.
- Lenguaje compatible con el sistema de gestión de base de datos MySQL, entre otros.

Existen diversas extensiones para este lenguaje. En nuestro proyecto utilizamos la **extensión 5** en su versión 5.5.8, instalada en **Raspberry Pi**, debido a los siguientes motivos:

- **Optimización con MySQL**: PHP 5 incluye nuevas funciones que mejoran el tiempo de respuesta en accesos a base de datos de sistemas MySQL. El uso de MySQL ha sido el principal detonante a la hora de elegir la extensión de PHP.
- **Programación orientada a objetos (POO)**: nos permite el uso de librerías necesarias para la realización del proyecto basadas en POO con PHP.

Por último, los principales usos que le hemos dado a PHP son los siguientes:

- **Comunicación entre Raspberry Pi y Arduino**: Mediante la librería gratuita *php-serial* desarrollada por Rémy Sánchez, podemos enviar órdenes desde Raspberry Pi a Arduino de una manera sencilla.
- **Consultas a base de datos**: Con el uso de funciones en PHP, podemos almacenar y obtener datos de la base de datos en MySQL.

Figura 5.29: Logo de PHP.

ProjETSII

ProjETSII es una herramienta basada en Redmine para la **gestión de proyectos** académicos, realizados únicamente en la Escuela Técnica Superior de Ingeniería Informática de Sevilla.

Debido a estar familiarizados con dicha herramienta, hemos optado por su inclusión en el proyecto para llevar a cabo el reparto de tareas entre los integrantes del grupo, anotando las horas estimadas y dedicadas de cada tarea.

PuTTy

PuTTy es un software cliente para protocolos SSH, Telnet y Rlogin, muy popular en entornos Windows. Desarrollado por Simon Tatham, este programa de licencia libre nos permite conectarnos al servidor SSH de Raspberry Pi en las ocasiones que deseamos hacer uso de la línea de comandos.

Python

Python es un lenguaje de código abierto, interpretado, sin control del tipo de variables, caracterizado por su legibilidad y diseñado por Guido van Rossum.

Entre sus distintas versiones, nos hemos decantado por Python 2.7, pues se trata de la versión más **estable** y es totalmente **compatible** con los módulos utilizados, también en Python, que detallamos a continuación:

- ***Módulo MySQLdb***: Nos da la posibilidad de hacer consultas a la base de datos MySQL.
- ***Módulo pySerial***: Con este módulo podemos tener control sobre los puertos serie de Raspberry Pi con el fin de recibir comandos de confirmación y datos procedentes de Arduino. También podíamos realizar estas tareas mediante **PHP**, pero tenemos mayor experiencia con Python.
- ***Módulo string***: Nos permite tratar los comandos que Raspberry Pi recibe de Arduino.

Figura 5.30: Logo de Python.

Raspbian

Raspbian es uno de los sitemas operativos compatibles con el dispositivo Raspberry Pi. Se trata de una distribución Linux **libre** basada en el sistema Debian 7.0 desarrollada por una comunidad sin ánimo de lucro (web oficial) y optimizada para la arquitectura de Raspberry Pi.

Raspbian ofrece numerosas características que favorecen su uso. Es por ello que en el momento de decidir qué sistema debíamos instalar en Raspberry Pi, pensamos que la mejor opción sería trabajar con Raspbian. Los puntos en los que nos apoyamos para llegar a esa conclusión son los siguientes:

- **Software libre**: Nos ayuda a reducir costes del proyecto.
- **Actualizaciones continuas**: Frente a otros sistemas, Raspbian se actualiza con regularidad para sacar el mayor partido de Raspberry Pi.

- **Compatibilidad**: Raspbian es uno de los sistemas que ofrece mayor número de controladores para la comunicación entre Raspberry Pi y otros dispositivos, como por ejemplo el adaptador WI-FI.
- **Debian 7.0**: Al basarse en Debian Wheezy es posible instalar el servidor web y la base de datos sin problemas.

Figura 5.31: Logo de Raspbian.

Revit

Revit es un software propietario para el diseño de edificios desarrollado por Autodesk que brinda una mayor rapidez a la hora de realizar los proyectos que otros programas profesionales como AutoCAD.

Nos ha sido de gran utilidad a la hora de trasladar los planos de la maqueta en papel a formato digital, añadiendo así más profesionalidad al proyecto.

Figura 5.32: Logo de Revit.

Sublime Text

Sublime Text es un editor de código desarrollado por Jon Skinner que soporta un **gran número de lenguajes** y es por esto por lo que nos ha resultado imprescindible en nuestro proyecto. Aún siendo un programa de pago, podemos descargar su versión de evaluación que posee todas las funcionalidades que necesitamos y no tiene limitación de tiempo.

Figura 5.33: Logo de Sublime Text.

TeXstudio

TeXstudio es un entorno de desarrollo integrado para realizar documentos en LaTeX. Es el resultado de una extensión del popular editor Texmaker.

La utilización de LaTeX nos lleva a la elección de uno de sus editores para facilitar la tarea de escribir la memoria. Entre todos sus editores, hemos optado por TeXstudio por los siguientes motivos:

- **Software gratuito**.
- **Detección de errores**: Además de detectar errores léxicos y sintácticos, muestra las posibles soluciones que los corrigen.
- **Autocompletado de comandos de LaTeX**.
- **Acceso directo** a etiquetas de LaTeX y multitud de símbolos matemáticos que nos simplifica la labor de introducir fórmulas, unidades de medida, etc.
- **Coloreado** de palabras clave y reservadas pertenecientes a LaTeX.
- **Soporte para *scripts***.
- **Exportación de documento en formato PDF**.

Figura 5.34: Logo de TeXstudio.

TightVNC

TightVNC es un software gratuito para el control remoto de escritorio creado por Constantin Kaplinsky. Con este programa podemos conectarnos al escritorio de Raspberry Pi cuando deseamos usar la interfaz gráfica que nos ofrece dicho dispositivo.

Figura 5.35: Logo de TightVNC.

6 Diseño e implementación

6.1 Diseño

Desde un enfoque general podemos decir que el sistema que hemos propuesto para cumplir con los objetivos está basado en el modelo de **comunicación Maestro/Esclavo**. El dispositivo Raspberry Pi tiene el rol de Maestro pues se encarga de enviar órdenes, mientras que la placa Arduino se comporta como Esclavo, procesando y ejecutando las órdenes procedentes de Raspberry Pi. Aunque en este tipo de diseños la comunicación es descendente, en nuestro sistema la comunicación es también **ascendente**. Con ello conseguiremos que la maqueta informe al usuario de condiciones que se dan en ella como por ejemplo: estado de los ledes, humedad, temperatura, alarmas,etc. A continuación, mostramos una abstracción de la comunicación:

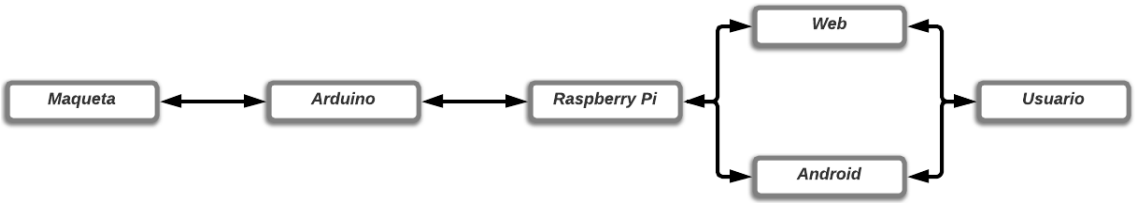


Figura 6.1: Abstracción de la comunicación del sistema domótico.

Como tecnología de interconexión entre Raspberry y Arduino hemos utilizado un cable **USB** tipo A tipo B administrando la transmisión de datos mediante el entorno de desarrollo de Arduino, Python y PHP. Por otro lado, para hacer posible la comunicación entre las aplicaciones, web y Android, y Raspberry Pi hemos usado la tecnología **WI-FI** y los protocolos TCP/IP.

Situándonos en cada nodo de este sencillo diagrama podemos apreciar que se comunica tanto con el nodo antecesor como con el sucesor. Para una mejor comprensión, pasaremos a explicar las comunicaciones descendente y ascendente que se establecen entre cada par de elementos que intervienen en el diagrama.

6.1.1 Comunicación descendente

Al comunicarnos de manera descendente perseguimos cumplir uno de nuestros **objetivos principales**: el diseño de un sistema domótico que permita el control de la maqueta.

- **Comunicación Usuario - Web**: El usuario interactúa con la aplicación web que proporciona Raspberry Pi. Mediante la interacción con elementos gráficos sencillos e intuitivos el Usuario podrá realizar las acciones que desee, por ejemplo, abrir una puerta.
- **Comunicación Usuario - Android**: De igual modo, el usuario también puede interactuar con la interfaz gráfica basada en Android para hacer uso de los servicios del

sistema domótico.

- **Comunicación Web - Raspberry Pi**: Las interacciones entre el usuario y los elementos de la aplicación web son traducidas en órdenes por dicha aplicación, y es la misma quien se encarga de enviarlas a Raspberry Pi.
- **Comunicación Android - Raspberry Pi**: Al igual que con las aplicación web, las interacciones entre el usuario y la aplicación Android son traducidas en órdenes y enviadas a Raspberry Pi por la propia aplicación Android.
- **Comunicación Raspberry Pi - Arduino**: Las ordenes enviadas procedentes de las aplicaciones web y Android son recibidas por Raspberry Pi, que posteriormente enviará a Arduino.
- **Comunicación Arduino - Maqueta**: Las órdenes que recibe Arduino de Raspberry Pi las procesa y ejecuta accionando así los componentes de la maqueta.

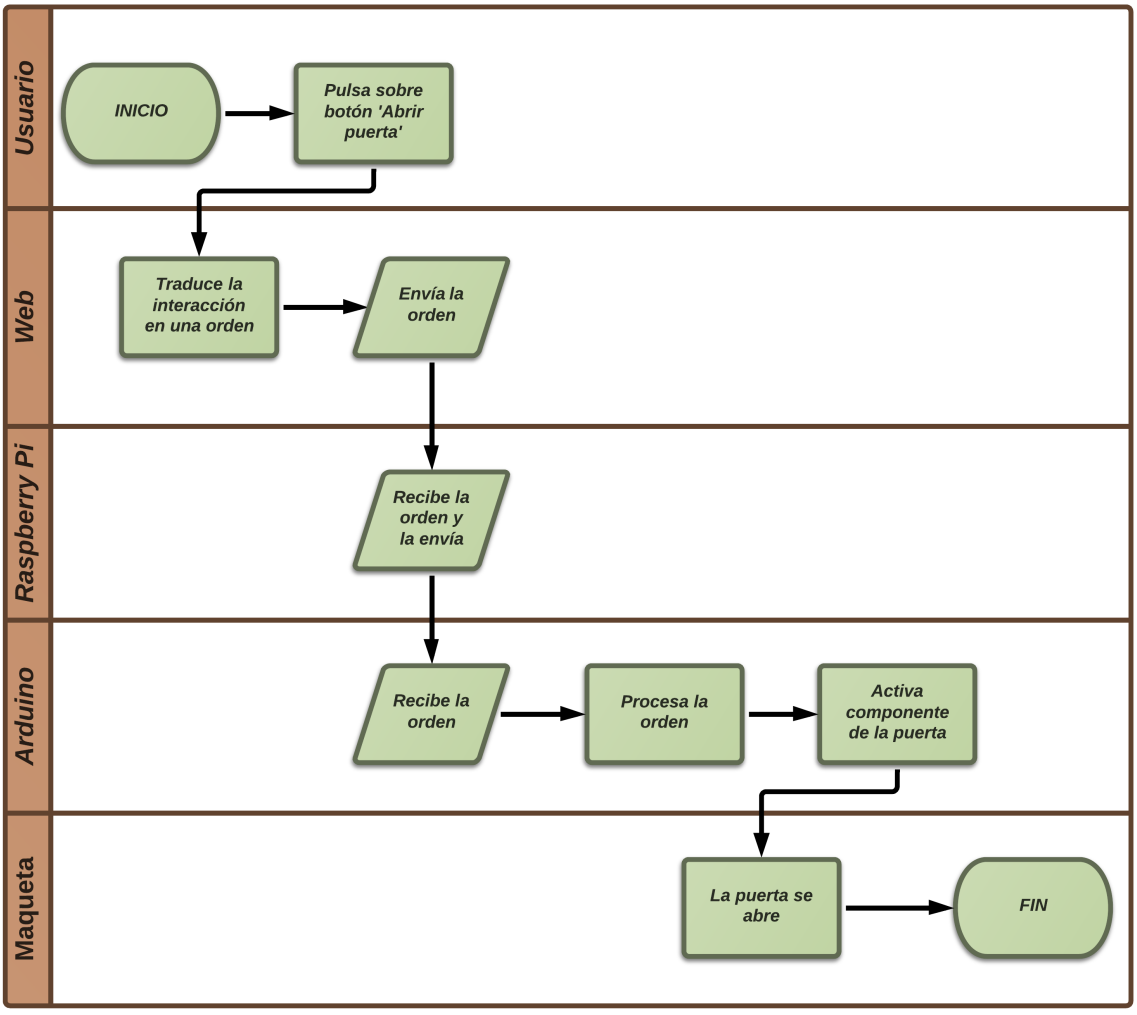


Figura 6.2: Diagrama de flujo de la comunicación descendente. Ej.: Apertura de una puerta.

6.1.2 Comunicación ascendente

Por medio de una comunicación ascendente buscamos satisfacer el **segundo objetivo principal**: el diseño de un sistema domótico que obtenga datos de eventos y elementos climáticos que tienen origen en la maqueta.

- **Comunicación Maqueta - Arduino**: Los sensores recogen datos sobre sucesos y elementos climáticos del entorno los cuales son enviados a Arduino.
- **Comunicación Arduino - Raspberry Pi**: Arduino se encarga de recibir los datos enviados por los sensores. Dependiendo del sensor que envíe los datos, Arduino realizará una de las siguientes opciones:
  - Enviar datos a Raspberry Pi.
  - Ejecutar una determinada acción y enviar notificación a Raspberry Pi.
- **Comunicación Raspberry Pi - Web**: Raspberry Pi recibe los datos procedentes de Arduino para almacenarlos y enviarlos a la aplicación web cuando ésta los requiera.
- **Comunicación Raspberry Pi - Android**: Al igual que en el caso de la aplicación web, Raspberry Pi envía los datos procedentes de Arduino a la aplicación Android tras el almacenamiento de los mismos cuando ella los requiera.
- **Comunicación Web - Usuario**: La aplicación web pide a Raspberry Pi, cada cierto tiempo, los datos que almacena procedentes de Arduino para transformarlos en información y mostrarla al usuario. El tiempo de las peticiones depende de la importancia de los datos que la aplicación web desea solicitar.
- **Comunicación Android - Usuario**: Del mismo modo que la aplicación web, la aplicación Android también pide a Raspberry Pi, cada cierto tiempo, los datos procedentes de Arduino para transformarlos en información y mostrarla al usuario. Este tiempo varía según los datos requeridos.

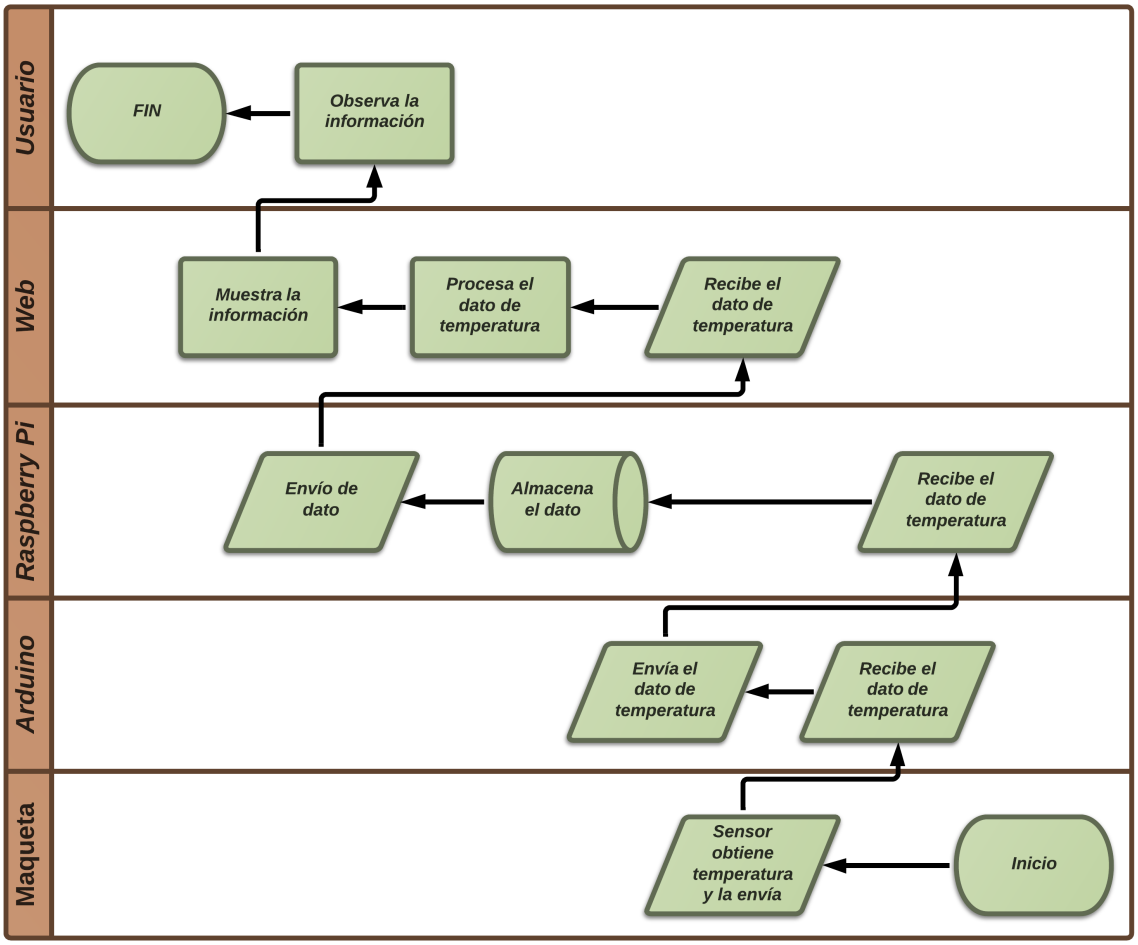


Figura 6.3: Diagrama de flujo de la comunicación ascendente. Ej.: Representar datos de temperatura.

6.1.3 Diseño interfaz gráfica

Desarrollaremos dos aplicaciones que servirán de interfaz gráfica de usuario. Con ellas podremos controlar la maqueta, tener acceso a la información recogida (temperatura, humedad, estado de la alarma, etc.) y demás servicios sin necesidad de tener conocimientos técnicos del sistema. Las aplicaciones son las siguientes:

- **Aplicación web**: Diseñada especialmente para dispositivos con pantallas grandes, como *tablets* u ordenadores, para una buen resultado del patrón Maestro/Detalle.
- **Aplicación Android**: Enfocada a dispositivos móviles basados en Android aprovechando el tamaño de sus pantallas.

Como podemos ver cada una está dirigida a un dispositivo distinto prestando especial atención al **tamaño de la pantalla** del mismo. En los dos siguientes apartados hablaremos del diseño de cada una de estas aplicaciones.

Aplicación web

El diseño de la aplicación web se apoya en el patrón **Maestro/Detalle** mostrando la información necesaria en un único espacio. El Detalle es una unidad de interacción que muestra información relacionada con el Maestro, siendo éste un objeto que determina

el contenido de las unidades de Detalle. A continuación, mostramos un prototipo de la página web identificando el patrón.

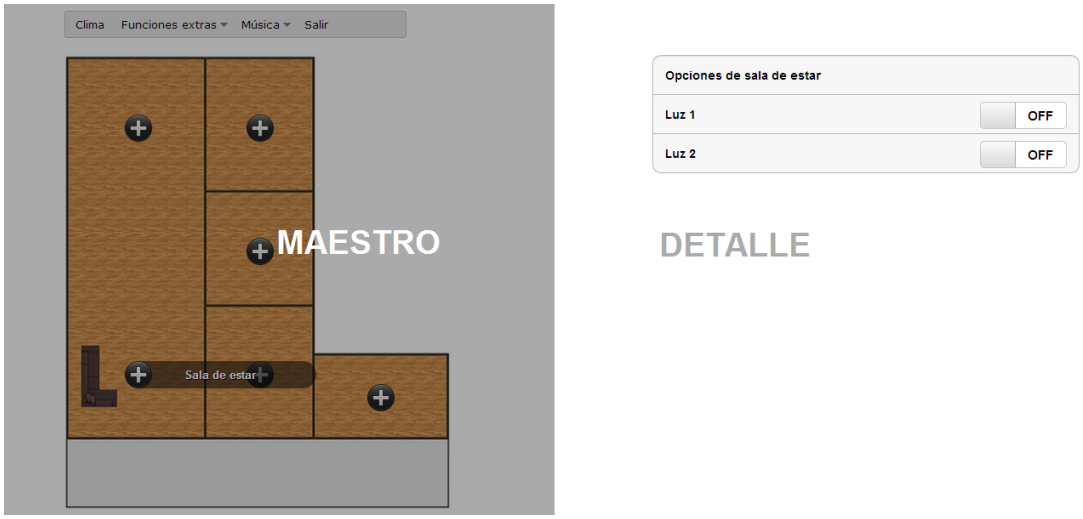


Figura 6.4: Maestro (parte sombreada) y Detalle del prototipo de la página web.

Como se puede ver en la imagen, a la izquierda (Maestro) encontramos los accesos directos a los servicios que ofrece el sistema, en forma de un plano de la maqueta y un menú horizontal. Cuando interactuamos con cada objeto del Maestro aparecerá sus opciones en el espacio asignado al Detalle **sin perder de vista** en todo momento al Maestro. Este comportamiento es una ventaja que afecta directamente al **usuario**: podrá acceder a todas las características de la interfaz sin la necesidad de navegar por diferentes páginas, es decir, todo lo que el usuario necesita para controlar el sistema está implementado en una sola página web y siempre está visible. Además, presenta otras ventajas que cumplen algunos de los requisitos de usabilidad como son: facilidad de aprendizaje del manejo de la interfaz y realizar una acción sin impedimentos.

Aplicación Android

La aplicación desarrollada en tecnología Android está basada en un diseño estándar de la misma. Tenemos **dos partes** claramente diferenciadas, llamadas *Activities* en esta tecnología. La *Activity* principal, o actividad principal, sirve de punto de entrada a la aplicación y consta de un formulario para autenticarnos contra el servidor. La segunda actividad de la aplicación es la que contiene toda la lógica de la misma y la que controlará, además, las comunicaciones de la aplicación.

Teniendo en cuenta la modularidad del sistema, hemos diseñado la aplicación de manera que cada habitación tenga su propia sección con sus elementos claramente localizados y diferenciables. Por suerte, Android dispone de varios patrones de navegación ampliamente usados y aceptados por la comunidad de desarrolladores. Hemos optado por una **navegación horizontal con pestañas**, su nombre en concreto es "*Tabbed Activity With Horizontal Swipe*". Podemos ver un esbozo de este patrón de navegación en la *figura 6.5*.

La tecnología Android permite el cambio de orientación del dispositivo adaptando la

interfaz de usuario a las nuevas dimensiones del espacio disponible. Sin embargo, hemos creído conveniente **desactivar** esta funcionalidad en nuestra aplicación por dos motivos: la aplicación queda mejor estructurada en una orientación vertical y evitamos elevar la complejidad de la implementación. Android está diseñado de forma que un cambio de orientación supone la **destrucción** de la instancia de la aplicación y la creación de otra nueva, por tanto, si deseamos realizar peticiones al servidor en segundo plano como es el caso, debemos tener en consideración la variedad de escenarios que podríamos encontrarnos.

El diseño de la comunicación de la aplicación lo hemos reducido a cómo enviamos datos al sistema y cómo los obtenemos. Android provee de implementaciones más o menos estándares para este cometido, sin embargo, dado que ya poseemos una implementación de la comunicación para la aplicación web, hemos decidido reutilizar una parte del código simplificando la implementación de la aplicación móvil desarrollada.

Para cerrar esta sección, recordamos que el proyecto tiene como uno de los requisitos que las aplicaciones de gestión del sistema domótico sean amigables y de una baja curva de aprendizaje, requisitos que han marcado la línea a seguir a la hora de desarrollar estas aplicaciones.

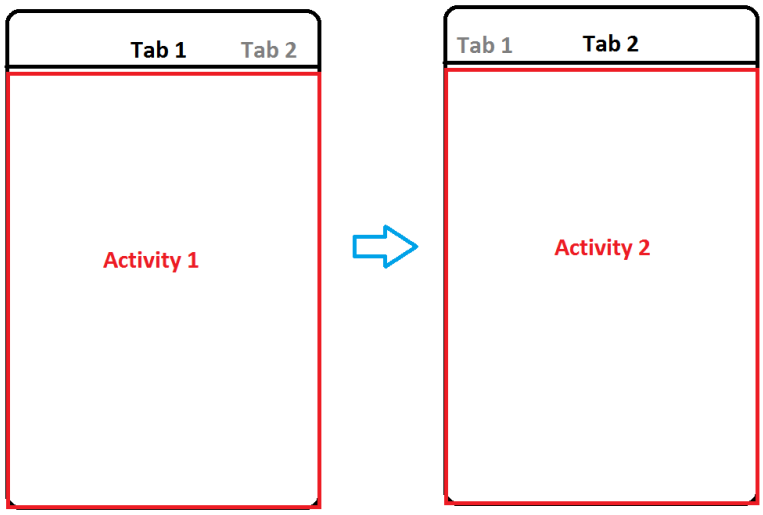


Figura 6.5: Patrón de Navegación.

6.2 Implementación

Procederemos a explicar el proceso de implementación para cada uno de los elementos (exceptuando Usuario) que participan en el diagrama de la comunicación *6.1: Abstracción de la comunicación del sistema domótico.*, anteriormente explicado, en sentido izquierda-derecha:

6.2.1 Maqueta

La maqueta consta de una cocina integrada con el salón, dos dormitorios, un baño y un garaje. Además, se ha reservado un espacio para los dispositivos del sistema domótico. Hemos utilizado dos tipos de tablón:

- **Conglomerado**: Empleado para construir la base de la maqueta. Tiene unas dimensiones de 95 cm x 95 cm x 1,6 cm (largo,ancho, altura). Nos da la suficiente resistencia para soportar las paredes y dispositivos electrónicos que albergará la maqueta.
- **MD**: Un tipo de conglomerado de menor densidad el cual nos ha servido para construir las paredes de la maqueta. La madera MD tiene una menor masa que el conglomerado de la base, haciendo que la masa total de la maqueta se incremente en pocas unidades. Los listones que simbolizan las paredes de la casa tienen las siguientes medidas:
  - Dos listones de 92 cm x 15 cm x 1 cm (largo, ancho, altura)
  - Dos listones de 76 cm x 15 cm x 1 cm (largo, ancho, altura)
  - Dos listones de 75 cm x 15 cm x 1 cm (largo, ancho, altura)
  - Dos listones de 22 cm x 15 cm x 1 cm (largo, ancho, altura)
  - Un listón de 25 cm x 15 cm x 1 cm (largo, ancho, altura)

Para unir las paredes entre ellas y la base, simplemente hemos utilizado adhesivo de montaje, evitando tener que atornillar los distintos tablones y , así, ahorrar tiempo. Además, hemos perforado algunos de estos tablones para hacer paso a los cables que conectan los componentes electrónicos y eléctricos a Arduino. A continuación, presentamos las vistas de la maqueta, la perspectiva isométrica y un foto real:

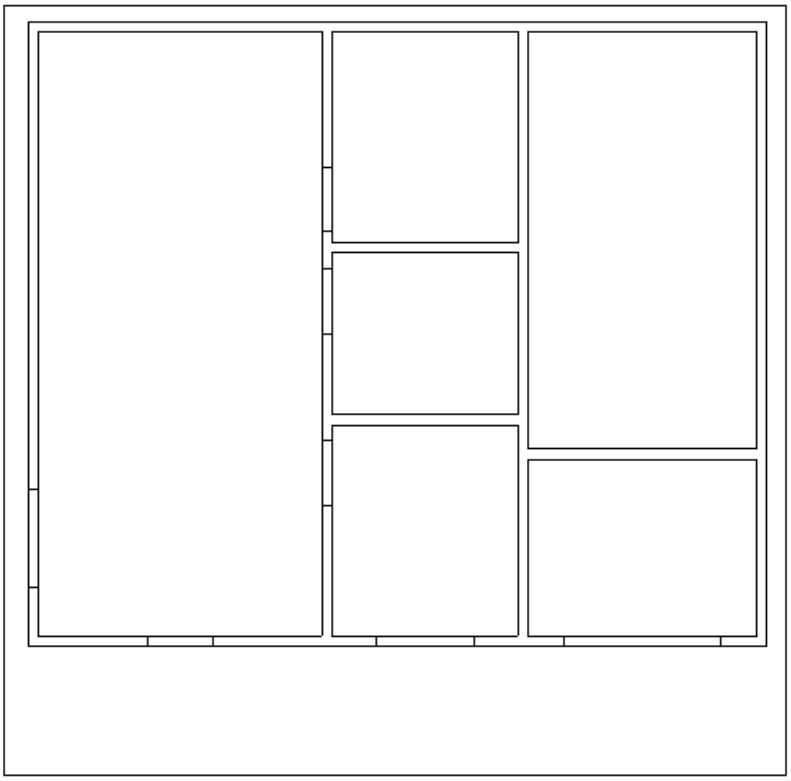


Figura 6.6: Planta de la maqueta.

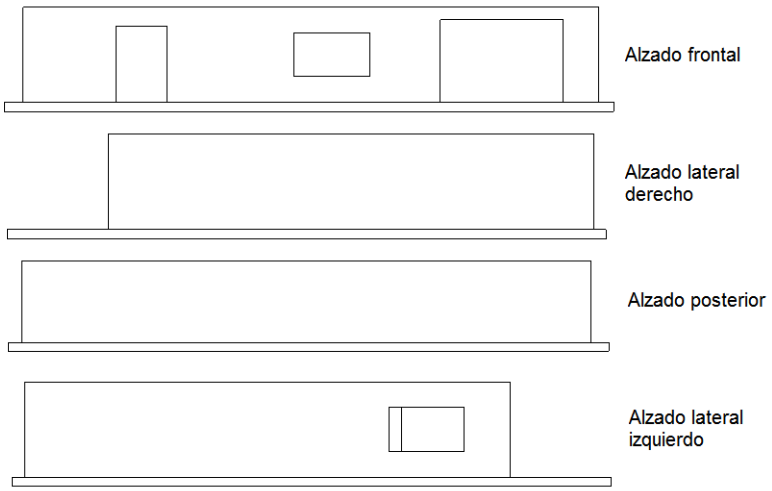


Figura 6.7: Vistas de la maqueta.

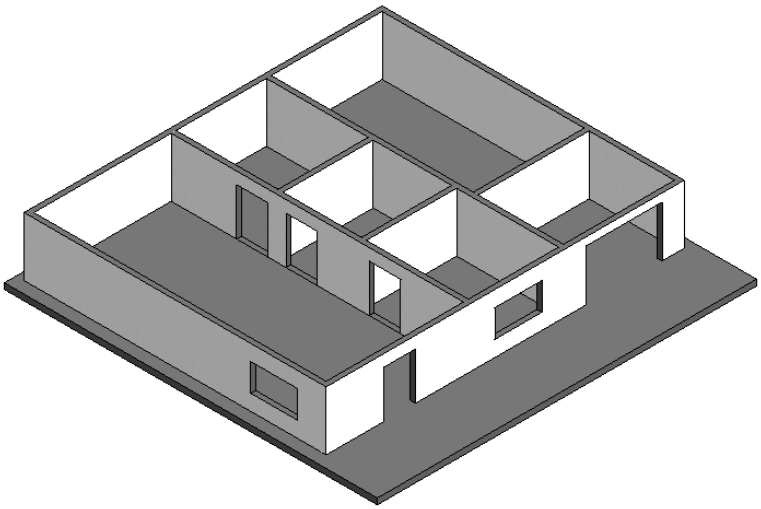


Figura 6.8: Perspectiva isométrica de la maqueta.

Figura 6.9: Más imágenes y vídeos en el DVD adjunto.

6.2.2 Arduino

En general, el trabajo de Arduino consiste en **enviar datos** obtenidos por sensores y **recibir y ejecutar órdenes** procedentes de Raspberry Pi a través de la comunicación serie entre ambos dispositivos mediante cable USB tipo A y tipo B. La manera en que se comunican ha dado lugar al **primer problema**, y es que Arduino se reinicia cada vez que recibe una orden perdiendo, así, el comando recibido. Ésto es debido a que cuando deseamos enviar un comando, establecemos comunicación con el puerto serie, enviamos el comando y cerramos la comunicación con el puerto serie. De esta forma fijamos unas buenas prácticas a la hora de manejar la comunicación, pero la consecuencia es que cuando cerramos la comunicación con dicho puerto, por defecto Arduino desactiva la línea DTR y activa el pin RESET provocando su reinicio. Para evitar ésto hemos colocado un condensador de 100μF entre los pines GND y RESET que absorbe el pulso eléctrico que activa el pin RESET evitando el reinicio.

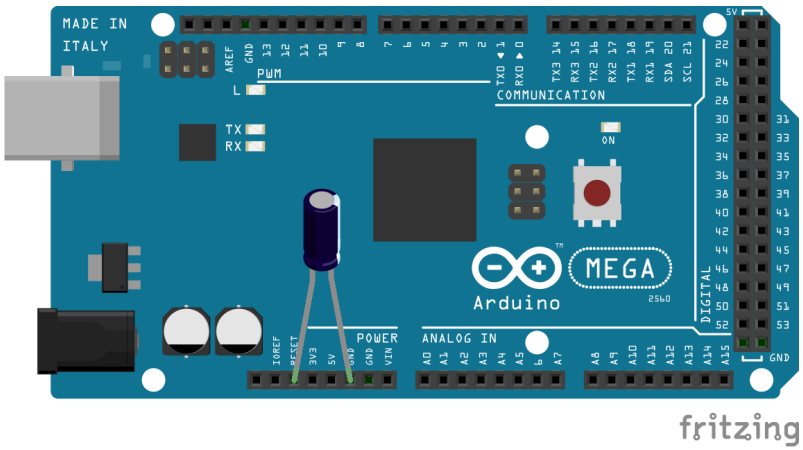


Figura 6.10: Condensador para evitar el reinicio de Arduino.

Como se ha comentado anteriormente, Arduino se encarga de recibir las órdenes en forma de comandos para después procesarlas y ejecutarlas. Puesto que Arduino sólo puede ejecutar un programa, todo el código necesario para la realización de las tareas correspondientes al dispositivo lo encontramos en el archivo ***Arduino.ino***.

Antes de explicar el proceso que realiza Arduino para llevar a cabo las órdenes procedentes de Raspberry Pi, vamos a explicar la **estructura** que presentan dichas órdenes con un sencillo ejemplo:

- Cuando deseamos encender la luz de una habitación de la maqueta mediante la interfaz gráfica, ya sea web o basada en el sistema operativo Android, la interfaz realiza una interpretación de la interacción entre nosotros y ella misma. Es decir, si pulsamos sobre un botón que nos ofrece la interfaz cuya función es la de apagado y encendido de la luz lo traducimos como una orden de encendido o apagado (según el estado del botón) que en realidad es un **comando** que la interfaz envía a Raspberry Pi, y éste envía a Arduino.

Los comandos que recibe Arduino de Rasbperry Pi son **cadenas de caracteres** (en mayúscula) y presentan la siguiente estructura:

- Tipo-Identificador-Valor*

Donde:

- **Tipo**: corresponde al tipo de componente que se desea accionar (led, motor,...)
- **Identificador**: nombre del componente que se desea accionar (led1,mo1,...) para poder distinguirlo entre el resto de componentes de su mismo tipo.
- **Valor**: establece lo que deseamos hacer con el componente (OFF, ON,...).

Como podemos observar, cada campo del comando va separado por un **guión**, '-', de esta forma podemos conocer el inicio y fin de cada uno de ellos. Además, al final cada comando añadimos un **asterisco**, '*', para indicar el fin de cada comando, más adelante hablaremos de ello. También hemos definido varios comandos que carecen de campo *valor* para comportamientos especiales, por ejemplo, apagar todas las luces.

Siguiendo con nuestro ejemplo, si deseamos encender la luz de una habitación cuyo led tiene como *identificador*, LED1, Arduino recibe el siguiente comando:

- `IL-LED1-ON*`

Las **ventajas** de implementar los comandos de esta manera son las siguientes:

- Sencillez y flexibilidad.
- Establecemos un patrón o guía que nos facilita futuras decisiones, por ejemplo: añadir un nuevo comportamiento.

También presenta algunas **desventajas**:

- Mayor tiempo para obtener el comando: Por cada carácter que tenga el comando Arduino tarda unas milésimas de segundo en obtenerlo (se ha establecido 8 ms), lo cual aumenta el tiempo de recepción del comando y, por consiguiente, aumenta el tiempo total del manejo del comando (tiempo que transcurre desde que comienza a obtener el comanda hasta que finaliza la ejecución del mismo).
- Debemos tener bien claro donde está situado cada componente puesto que en el comando no aparece a qué parte de la maqueta pertenece (cocina, sala de estar, etc.).

Con las siguientes tablas resumimos todos los comandos que recibe Arduino acompañados de una breve descripción:

Tipo	Identificador	Valor	Descripción
MO	MO1	UP	Abrir puerta del garaje
		DOWN	Cerrar puerta del garaje
MO	MO2	OP	Abrir puerta principal
		CL	Cerrar puerta principal

Cuadro 6.1: Comandos dedicados a control de motores.

Tipo	Identificador	Valor	Descripción
VE	V1	ON	Activar ventilación
		OFF	Desactivar ventilación

Cuadro 6.2: Comandos dedicados a control de la ventilación.

Tipo	Identificador	Valor	Descripción
AL	A1	OFF	Desactivar alarma

Cuadro 6.3: Comandos dedicados a desactivar la alarma.

Tipo	Identificador	Valor	Descripción
IL	LED1	ON	Encender luz de la cocina
		OFF	Apagar luz de la cocina
IL	LED2	ON	Encender luz 1 de la sala de estar
		OFF	Apagar luz 1 de la sala de estar
IL	LED3	ON	Encender luz 2 de la sala de estar
		OFF	Apagar luz 2 de la sala de estar
IL	LED4	ON	Encender luz del baño
		OFF	Apagar luz del baño
IL	LED5	ON	Encender luz del garaje
		OFF	Apagar luz del garaje
IL	LED6	ON	Encender luz del dormitorio 2
		OFF	Apagar luz del dormitorio 2
IL	LED7	0 a 255	Encender / apagar gradualmente luz del dormitorio 1
IL	ALL	-	Encender todas las luces de las habitaciones
IL	ANY	-	Apagar todas las luces de las habitaciones

Cuadro 6.4: Comandos dedicados a la iluminación.

A continuación, detallamos los procedimientos que sigue Arduino para la **recepción del comando** y el **proceso de ejecución** del mismo, así como el proceso para enviar datos de sensores a Raspbery Pi. Por último, dedicaremos un apartado para **funcionalidades extras** que realiza Arduino.

Recepción y ejecución del comando

Para que Arduino reciba correctamente el comando, el dispositivo que lo envíe debe transferir a la misma velocidad que Arduino lea por el puerto serie. Esta **velocidad de transmisión** la definimos con la función *Serial.begin()* incluida en el software oficial de Arduino. Como hemos visto veces anteriores, esta función recibe como parámetro la velocidad de transmisión representada en baudios o bits por segundo. Normalmente, usamos un valor de 9600 baudios (establecido por defecto).

Tras seleccionar la velocidad de transmisión, debemos **obtener cada carácter** del comando mediante la función Serial.read() de Arduino. Para hacerlo más sencillo hemos definido la **función *getCommand()***, que recibe dos parámetros: un array donde se almacenará el comando y un entero que nos ayudará a almacenar cada carácter, excepto el

carácter asterisco ('*'), en una posición del array distinta. Para obtener el comando cuando lo enviamos a Arduino usamos la función *Serial.available()* que devuelve 0 si no hay bytes en el buffer de lectura, o el número de bytes que hay en dicho buffer. Como máximo el tamaño del buffer de lectura del puerto serie es de 128 bytes de los cuales, 127 bytes están disponibles para los comandos, espacio suficiente para un comando sabiendo que cada carácter es un byte.

Por último, haremos un pequeño inciso sobre el **carácter asterisco** al final de cada comando para indicar su final. En realidad, para acciones sencillas, como la apertura de una puerta, no necesitamos este carácter, ya que Arduino es capaz de recibir este comando vaciando el buffer de lectura lo suficientemente rápido como para estar preparado para leer otro comando, por ejemplo encender un led. Pero sí necesitamos este carácter para implementar otros comportamientos los cuales necesitan del envío de un gran número de comandos consecutivos que se encuentran al mismo tiempo en el buffer.

Tras obtener el comando, nos disponemos a almacenar cada **campo** que lo compone (*tipo*, *identificador* y *valor*) en diferentes arrays mediante la función *strtok_r()* propia del lenguaje C. Posteriormente, nos apoyamos en sentencias condicionales *if-else if-else* para comparar los distintos campos del comando con los posibles valores que pueden tomar hasta llegar a la acción que deseamos que realice Arduino.

Envío de datos de sensores

Arduino obtiene datos de sensores colocados en la maqueta cada cierto tiempo y los envía a Raspberry Pi para posteriormente procesarlos. Para manejar de una forma sencilla estos datos con Raspberry Pi hemos decidido enviarlos mediante **cadenas de caracteres** que presentan la siguiente estructura:

- Sensor-Valor

Donde el campo **Sensor** indica el identificador, y el campo **Valor** indica el dato que recoge el sensor.

El problema que conlleva enviar datos cada cierto tiempo es que Arduino puede enviar al mismo tiempo uno o más de estos datos y una confirmación de orden ejecutada **concatenando los envíos**. Por ejemplo, si Arduino envía una confirmación de led encendido, `CM-IL-LED1-ON`, y un dato recogido sobre la temperatura, `TM-20`, Raspberry Pi recibe `CM-IL-LED1-ONTM-20`. Para evitar esto hemos decidido que antes de enviar datos de los sensores y confirmaciones de ordenes ejecutadas, Arduino debe esperar a que el buffer de envío esté vacío mediante la función *Serial.flush()* propia del lenguaje Arduino.

Por último, adjuntamos la tabla de los comandos que envía Arduino a Raspberry Pi con el propósito de transmitir los datos que recogen los sensores:

Sensor	Valor	Descripción
`TM`	`XX`	Dato de temperatura
`HM`	`XX`	Dato de humedad relativa

Cuadro 6.5: Comandos dedicados al envío de datos de sensores. (*) XX indica cualquier valor registrado por el sensor.

Funcionalidades extras

Además de recibir y ejecutar órdenes, Arduino también tiene la tarea de enviar ordenes y datos a Raspberry Pi. Para ello nos valemos de las funciones *Serial.print()* y *Serial.println()* las cuales envían por el puerto serie el argumento que reciben de tipo array de caracteres. A continuación pasamos a describir otras funcionalidades que realiza Arduino a parte de las ya vistas:

- **Confirmación de orden ejecutada**: Tras ejecutar un comando que afecte a pines conectados a componentes que se encuentran en cierto estado (encendido/apagado, abierto/cerrado,etc.), Arduino reenvía dicho comando a Raspberry Pi pero añadiendo al principio otro campo más, **`CM`**, para indicarle a Raspberry Pi que es un comando de confirmación. Por tanto, la respuesta de confirmación en el caso de encender un led sería: `CM-IL-ID-ON`.

- **Inteligencia artificial**: Debido a que empleamos varios sensores podemos otorgar a Arduino la responsabilidad de tomar sus propias decisiones. Por ejemplo, Arduino puede activar la ventilación a una determinada temperatura mediante el sensor DTH11 o cerrar la puerta del garaje una vez aparcado el coche en él por medio del sensor HC-SR04.

- **Notificación de arranque**: Cuando iniciamos o reiniciamos Arduino, los pines digitales son desactivados afectando a ciertos componentes con su consecuente cambio de estado. En un principio los componentes que se verán afectados serán ledes y alarmas. Para hacer presente los cambios en la interfaz gráfica, Arduino envía el comando **`CM-IL-RESET`** a Raspberry Pi. Si nos fijamos, hemos reutilizado el campo `CM` para ahorrar código y hacerlo más sencillo. Este comando notifica a Raspberry Pi que hemos puesto en marcha Arduino.

  Dado que sólo es necesario enviar el comando cada vez que Arduino inicia, lo hemos implementado en la función *setup()* de Arduino que es ejecutada únicamente una vez en el momento en el que Arduino se inicia.

- **Notificación de eventos**: A través del micrófono de contacto, Arduino nos avisará de que existe una alta concentración de gas combustible detectada por el sensor MQ-2. Cuando dicho sensor detecte una cantidad mayor a 2000 ppm[1] el micrófono actuará de alarma y emitirá un pitido a 3 kHz[2] cada 0,75 segundos para alertar al usuario.

6.2.3 Raspberry Pi

Como ya hemos mencionado, Raspberry Pi es un minicomputador lo cual hace que sea más sofisticado a nivel de hardware y software que Arduino. Debido a ello, este dispositivo desempeña numerosas tareas en nuestro proyecto de las cuales destacamos:

- **Enviar órdenes a Arduino**

[1] Este valor es sólo un umbral de demostración. Cada país se rige por unas leyes y normas que determinan la cantidad de gas y el tiempo de exposición que supondrán perjudiciales a las personas.

[2] Se ha establecido este valor debido a que el rango de frecuencia entre 1 kHz y 3 kHz es más sensible para el oído humano.

- **Recibir comandos de confirmación y datos de sensores enviados por Arduino**

Pasamos a explicar el proceso de implementación para llevar a cabo las tareas que acabamos de comentar, además de algunas funcionalidades extra que aborda este dispositivo.

Enviar órdenes a Arduino

Raspberry Pi recibe órdenes del usuario a través de las aplicaciones web y Android. Posteriormente dichas órdenes serán transmitidas a Arduino. El proceso por el cual Raspberry Pi recibe y envía las órdenes está implementado en el archivo ***toArduino.php*** alojado en el servidor Apache del dispositivo mencionado. A continuación, explicamos dicho proceso:

1. **Recepción de una orden**: Cuando alguna aplicación envía una orden al servidor Apache, éste la recibe consultando la variable *command* almacenada en el array *$_POST* de PHP.

2. **Envío de una orden**: Si la variable *command* está definida, establecemos la conexión con Arduino y procedemos a enviar la orden. Para ello nos hemos apoyado en la clase ***php-serial*** con la cual hemos definido los parámetros de conexión:

   - Puerto: ttyACM0.
   - Tasa de baudios: 9600.
   - Bit de paridad: Ninguno.
   - Longitud de caracteres: 8 bits.
   - Bits de parada: 1 bit.

   La misma clase nos ha ayudado también en el envío de la variable *command* a Arduino completando el proceso de envío de la orden. Utilizado las funciones *deviceOpen()* y *deviceClose()* hemos gestionado la conexión con Arduino, y mediante la función *sendMessage()* hemos enviado la variable *command*.

Recibir comandos de confirmación y datos de sensores enviados por Arduino

Raspberry Pi almacena los comandos de confirmación de orden ejecutada procedentes de Arduino, y de igual modo, guarda los datos que recogen los sensores para procesarlos posteriormente. Para lograr estos objetivos hemos implantado una base de datos **MySQL** en la que hemos definido las siguientes tablas:

- **alarms**: Almacena la información del estado de las alarmas (si se encuentra activadas o desactivadas).

- **humidity**: Almacena los datos correspondientes a la humedad y el momento en el que se han obtenido.

- **statepin**: Almacena los comandos de confirmación de orden ejecutada que reenvía Arduino. La tabla contiene los atributos correspondientes a los campos que forman el comando: *tipo*, *identificador* y *valor*.

- **temperature**: Almacena los datos correspondientes a la temperatura y el momento en el que se han obtenido.
- **users**: Almacena las credenciales de usuario necesarias para acceder a la interfaz gráfica.

Mostramos, a continuación, el modelo relacional de la base de datos:

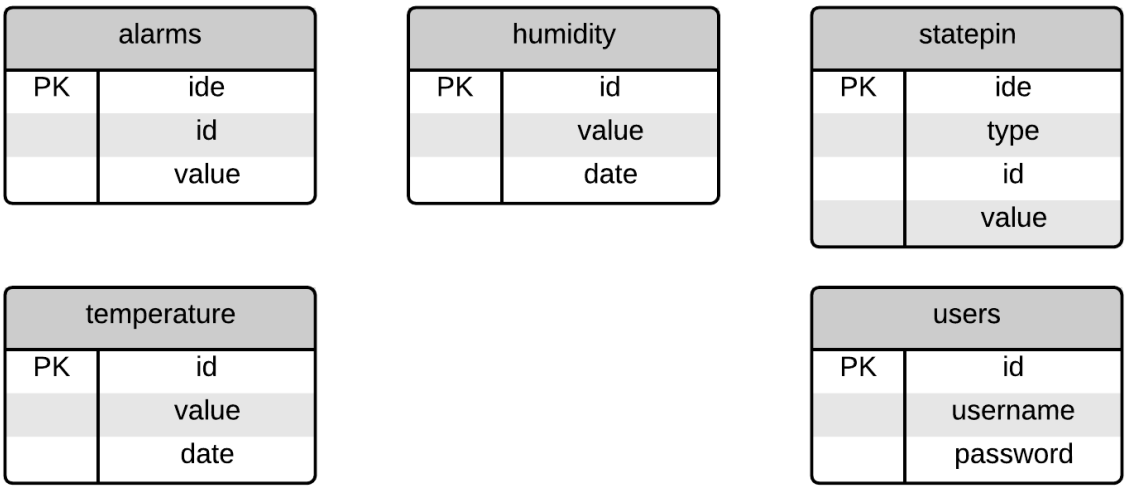


Figura 6.11: Modelo relacional de la base de datos.

El proceso que sigue Raspberry Pi para almacenar la información que llega de Arduino está definido en el archivo *arduinoToMySql.py* desarrollado en Python que se ejecuta al iniciar Raspberry Pi. El proceso se compone de los siguientes pasos:

1. **Recepción de datos**: Mediante la librería *pySerial* establecemos conexión con Arduino a la espera de recibir datos de algún tipo. Lo que envíe Arduino, Raspberry Pi lo almacena en un array mediante la función *readline()* de la librería *pySerial*.
2. **Identificación de datos recibidos**: Raspberry Pi debe distinguir si se trata de un comando de confirmación o de un dato de algún sensor. Como ya hemos comentado en otros apartados, los comandos de confirmación siguen la estructura *CM-Tipo-Identificador-Valor* y los datos de los sensores se envían de la forma *Sensor-Valor*, por lo tanto, simplemente tenemos que separar los campos delimitados por el carácter '-' y determinar si es el campo *CM* o el campo *Sensor* con una estructura condicional *if-elsif* en Python.
3. **Comunicación con la base de datos**: Tras identificar la cadena de caracteres recibida, establecemos comunicación con la base de datos MySQL mediante la librería *MySQLdb* para Python. A través de consultas a la base de datos almacenaremos los campos que nos interese de la cadena en la correspondiente tabla.

Funcionalidades extras

Pasamos a definir otras labores que lleva a cabo Raspberry Pi:

- **Actualización de la tabla *statepin* al inicio o reincio de Arduino**: En el apartado de Arduino comentamos que dicho dispositivo envía la notificación de arranque

`CM-IL-RESET` para informar que acaba de iniciar o reiniciar desactivando los pines digitales dando lugar a que los ledes se desactiven. Raspberry Pi se encarga de recibir esta notificación y actualizar la tabla *statepin* de la base de datos, estableciendo a aquellas filas pertenecientes a los ledes el valor `OFF` en la columna *value*. Para recibir esta notificación hemos usado el mismo procedimiento que el utilizado para recibir los comandos de confirmación, desarrollado en el archivo *arduinoToMySql.py*.

- **Eliminar registros de datos innecesarios**: Debido a que mostramos al usuario datos de temperatura y humedad durante el día, debemos eliminar valores registrados anteriormente puesto que además de no ser necesarios, agilizamos la búsqueda y obtención de resultados de la base de datos. Hemos establecido que la base de datos elimine automáticamente los registros innecesarios de las tablas correspondientes a los datos que mostramos mediante la sintaxis `CREATE EVENT` de MySQL. Por tanto, cada día a las 00:00:00 eliminará los datos innecesarios.

- **Login para interfaz gráfica**: Raspberry Pi almacena en la base de datos el usuario y contraseña que tenemos que usar para poder acceder a la interfaz gráfica. En la aplicación web nos apoyamos de *array* global *$_SESSION* de PHP para almacenar una variable de autorización la cual nos permitirá entrar en la página web principal, *main.php*.

- **Servidor de streaming de audio**: Mediante el *widget* jPlaylister hemos añadido un servidor para poder escuchar música desde nuestros ordenadores u otros dispositivos personales. Podemos reproducir los archivos (mp3, mp4, ogg, webm y wav) desde la aplicación web.

6.2.4 Aplicación web

La aplicación web que proporciona Raspberry Pi está almacenada en el servidor Apache. Para acceder a ella, primero hemos implementado un página web de acceso que corresponde con el archivo *index.php*. En este archivo hemos definido un pequeño formulario de *login* que hay que cumplimentar para poder acceder a la página web principal ***main.php***.

El diseño se ha implementado en HTML5 y JavaScript. Por medio de etiquetas *div* hemos organizado y distribuido cada parte de la página web, estableciendo qué elementos pertenecen al **Maestro** o al **Detalle**. En el fichero ***main.js*** y con los métodos *click()*, *show()* y *hide()* de JavaScript hemos implementado la filosofía Maestro/Detalle definida en la sección *6.1 Diseño*. Ademas, nos hemos apoyado en el lenguaje CSS3 y la librería jQuery para darle mayor riqueza visual y dinamismo a los elementos gráficos que componen la web.

La aplicación web tiene como **principal objetivo** facilitar al usuario la tarea de controlar la maqueta mediante una interfaz gráfica sirviendo de capa de abstracción de la funcionalidad del sistema domótico. Pasamos entonces a explicar la implementación del proceso que hace realidad dicho objetivo:

- A cada **elemento gráfico** (botones, barras de desplazamiento, etc.), que requiera controlar una parte de la maqueta, le hemos asignado un comando equivalente

a la orden que representa el elemento. Luego, si tenemos un botón que se encarga de apagar la luz de una habitación, a dicho botón le asignamos el comando `IL-LED1-ON`, donde *IL* define que el componente a accionar es de *tipo* iluminación, *LED1* corresponde al identificador del componente, y *ON* su comportamiento como ya mencionamos en el apartado de *Diseño*.

- La asignación de comandos la hemos realizado mediante lenguaje **JavaScript**, y se encuentra implementada en el archivo *main.js* convirtiéndose en el **intérprete de comandos** para la aplicación web .

- El comando correspondiente al elemento gráfico con el que interactuamos lo guardamos en la variable *command* y la **transmitimos** mediante POST con la técnica **AJAX** al archivo en lenguaje **PHP**, *toArduino.php*, alojado en el servidor Apache de Raspberry Pi.

- Si un determinado elemento gráfico requiere del envío sucesivo de comandos, los transmitiremos sin la necesidad de cerrar la conexión entre Arduino y Raspberry Pi por cada comando que se envíe, mejorando de esta manera la velocidad de respuesta.

Las funciones implementadas en la página web principal van mucho más allá del control de la maqueta. Pasamos a definir estas funciones disponibles para el usuario:

- **Actualización de los elementos gráficos de la aplicación web**: El propósito por el que hemos almacenado los comandos de confirmación en la base de datos MySQL es conseguir un correcto funcionamiento de la interfaz gráfica, es decir, si encendemos un led mediante un botón, éste cambia para notificar al usuario de su acción. Si actualizamos la página web, el botón indica que el led está apagado, pero en realidad el led sigue encendido. Mediante una consulta en la base de datos usando la técnica AJAX obtenemos el estado del componente que es accionado por ese botón, a continuación usamos la librería jQuery para actualizar el estado del botón según el resultado de la consulta. Este proceso lo ejecutamos cada vez que accedamos a la página web.

  Es lógico pensar que puede haber **varios usuarios** usando las aplicaciones web y Android al mismo tiempo. Esto significa que los cambios que uno de los usuarios realiza deben constar en las instancias de las aplicaciones del resto de usuarios. En el caso de la página web, el proceso de actualización de los elementos gráficos se realiza periódicamente **cada 30 segundos** de manera asíncrona, además del momento en el que accedemos a ella.

- **Control de la sesión**: Mediante un sencillo botón basado en jQuery y el lenguaje PHP el usuario puede salir de la página web cerrando la sesión que tenía abierta. Si el usuario está ausente durante unos minutos mientras que la sesión esté abierta, ésta se cerrará automáticamente.

- **Gestor de archivos**: Por medio del *widget* elFinder 2.0 hemos implementado un gestor de archivos en nuestra aplicación web con el fin de permitir al usuario subir canciones para reproducirlas posteriormente con el reproductor de música. Hemos

establecido 1 GB de espacio de almacenamiento disponible en Raspberry Pi dedicado a las canciones que el usuario suba. Este espacio se ha implementado mediante el uso cuotas de disco explicado en el apéndice *Configuración de Raspberry Pi*.

Para un uso práctico del *widget*, hemos tenido que modificar la configuración del módulo PHP del servidor Apache, que nos restringía subir archivos de más de 2 MB desde el gestor. La modificación se ha realizado sobre el archivo *php.ini*, en concreto, en las lineas `upload_max_filesize` y `post_max_size`, estableciendo un valor de 1 GB, de esta forma lo igualamos al límite máximo de almacenamiento para el usuario.

- **Información sobre el entorno**: Hemos implementado una serie de gráficos de líneas basadas en la librería jQWidgets que muestran datos del entorno, como temperatura y humedad relativa, recogidos durante el día. Estos gráficos son actualizados cada minuto mediante la función *setInterval()* de jQuery e informan al usuario de los elementos climáticos durante el día, así como los actuales. Las gráficas se alimentan de los datos recogidos del resultado de las búsquedas en la base de datos MySQL en formato JSON mediante el uso de AJAX. Junto a los gráficos de líneas, hemos añadido elementos textuales que muestran las últimas condiciones del entorno.

- **Reproductor de música**: Hemos añadido una interfaz gráfica para reproducir la música que almacenemos en el servidor de streaming de audio (más adelante hablaremos de él). Para hacer posible dicha interfaz nos hemos apoyado en la librería jPlaylister.

- **Notificación de eventos**: Cuando deseamos comunicar al usuario de que ha ocurrido algún imprevisto y éste requiere de su atención, utilizamos la ventana de alerta del navegador. Esta ventana paraliza el hilo de ejecución de la aplicación web y no permite continuar usándola hasta que no se cierre dicha ventana. De esta manera nos aseguramos de que el usuario se percata de que ha ocurrido algún problema grave.

  En un principio, esta característica se ha reservado sólo para la alarma. La aplicación web consulta cada 5 segundos la base de datos de Raspberry Pi de forma dinámica mediante AJAX para comprobar si la alarma se ha activado. En caso afirmativo, mostramos el mensaje de alerta: *¨ Alarma activada. Posible fuga de gas. Pulse aceptar o cierre la ventana para desactivarla"*. El tiempo de aparición de esta alerta es de 0 a 5 segundos.

6.2.5 Aplicación Android

La aplicación desarrollada en Android comparte objetivos con su homóloga, anteriormente explicada. Es por ello, que la implementación reutiliza componentes de la aplicación web, aunque este tema será explicado más adelante en este mismo capítulo. La aplicación está desarrollada en lenguaje 100 % nativo, *Java-Android*, y no hemos hecho uso de ninguna librería de terceros. A continuación emplearemos términos técnicos relativos al sistema operativo Android. Para entrar más en detalle acerca de estos términos

podemos acudir a la *documentación oficial de Android*.

En primer lugar, al lanzar la aplicación veremos una ventana de *Login*. La implementación que la hace posible es muy simple y el mismo **Android Studio** (el IDE para desarrollo android de Google) contiene una plantilla para este tipo de pantalla. Se trata de una Activity en cuyo **Layout** (el archivo *XML* que define la interfaz de usuario) encontramos dos componentes **EditText** para la inserción de texto. En la plantilla proporcionada contamos con métodos para la validación de los campos introducidos y una clase interna que hereda de la *AsyncTask()* y que hemos usado para construir un objeto que nos permita hacer peticiones al servidor en segundo plano. Esto requiere una aclaración: desde Android versión 3.0 no se pueden realizar tareas de larga duración en el mismo hilo de ejecución que la interfaz de usuario porque podría ocasionar la congelación de la aplicación. Para esto, los desarrolladores de Android crearon un objeto enfocado a realizar estas tareas pesada en otro hilo de ejecución, el mencionado anteriormente **AsyncTask()** que usaremos a lo largo de toda la implementación para las comunicaciones con el servidor.

En el capítulo de diseño hemos hablado de un **patrón de navegación**, que no es más que una forma de plantear el flujo de trabajo a través de la aplicación. Si bien existen varios que se adecuan a nuestras necesidades, hemos decidido prescindir del *Master-Detail* que implementa la aplicación web y optar por uno nativo en Android. El elegido en cuestión es el "*Tabbed Activity With Horizontal Swipe*" que consiste en una navegación basada en *tabs* o pestañas y la posibilidad de deslizar con el dedo (movimiento *swipe*) para acceder a la siguiente. La implementación de este patrón se basa en una sola *Activity* y múltiples ***Fragments*** o fragmentos, que como su propio nombre indica, sirven como *Activities* parciales. Aprovechando la naturaleza de estos fragmentos podemos encapsular la lógica de aplicación y sustituirla o componerla con más funcionalidades, muy útil en tablets. De esta forma, hemos implementado la Activity padre que sostendrá la aplicación y la funcionalidad de cada parte de la maqueta la hemos encapsulado en fragmentos distintos. Hemos de tener en cuenta que esta implementación requiere de mecanismos especiales para comunicar la Activity principal con los fragmentos que dependen de ella.

Una vez explicado el patrón de navegación principal podemos entender como está estructurada la aplicación. Hemos definido nuestros propios botones para la interfaz de usuario a partir del objeto *ToggleButton* de Android. Este objeto tiene la peculiaridad de que alberga dos estados en sí mismo y podemos consultar si está activado o no fácilmente. Tanto como para los botones referentes a la iluminación del sistema como para los mecanismos motorizados hemos definido este tipo de botones, personalizando las apariencias de sus estados. Del mismo modo que la aplicación web mandaremos un comando de texto al servidor según el estado de estos botones.

Otros recursos gráficos que hemos empleado son la decoración de la *ActionBar* o barra superior de navegación y la de las pestañas. Estos recursos gráficos han de ser imágenes estáticas existentes en las carpetas de recursos de la aplicación y han de ser compuestos a través de ficheros *XML*.

A continuación nos adentraremos en la implementación de los mecanismos de comunicación de la aplicación con el servidor. Como hemos mencionado anteriormente la

comunicación con el servidor debe hacerse con un objeto *AsyncTask* para no bloquear el hilo principal de ejecución de la aplicación y, además, hemos querido reutilizar parte del código de la aplicación web. Con estos prerequisitos hemos creado un objeto propio que herede de la clase AsyncTask y que se ajuste a nuestras necesidades. Este tipo de objetos tienen dos métodos principales que describiremos a continuación:

- **doInBackground()**: es el método que se ejecutará en segundo plano y donde debemos encapsular toda la lógica de la aplicación como peticiones a servidores u otras tareas de posible larga duración.
- **onPostExecute()**: es el método que se ejecutará justo despues de que *doInBackground()* finalice. Aquí es donde debemos de localizar la lógica de actuación con respecto a los resultados de las peticiones.

Sin adentrarnos demasiado en el lenguaje Java-Android esto es, a grandes rasgos, lo que debemos de saber. Por lo tanto, incluimos a continuación nuestra clase *java* "**SendCommandTask**"

Listing 6.1: Detalle del objeto SendCommandTask..html

```
package com.us.domotica.app.util;

import android.content.Entity;
import android.os.AsyncTask;
import android.util.Log;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;

import java.io.BufferedReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

/**
 * Created by Miguel on 10/05/14.
 */
public class SendCommandTask extends AsyncTask<Void,Void, HttpEntity> {

    /*
     mandatory parameters for the server
     serverFile as toArduino.php
     command as MO-MO1-UP
     */
    private String serverUrl;
    private String command;



    public SendCommandTask(String serverUrl, String command){
        this.serverUrl = serverUrl;
        this.command = command;
    }

    @Override
    protected HttpEntity doInBackground(Void... voids) {
        BufferedReader inBuffer = null;
        HttpEntity entity = null;
        try {
            HttpClient httpClient = new DefaultHttpClient();
            HttpPost request = new HttpPost(serverUrl);
            List<NameValuePair> postParameters = new ArrayList<NameValuePair>();
            postParameters.add(new BasicNameValuePair("command", command));

            UrlEncodedFormEntity formEntity = new UrlEncodedFormEntity(
                    postParameters);

            request.setEntity(formEntity);

            HttpResponse response = httpClient.execute(request);
            entity = response.getEntity();
```

```
        } catch(Exception e) {
            Log.d("App Exception: ",e.getMessage());

        } finally {
            if (inBuffer != null) {
                try {
                    inBuffer.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return entity;
    }

    @Override
    protected void onPostExecute(HttpEntity entity) {

    }
}
```

Como pretendemos reutilizar los archivos PHP creados para la aplicación web, necesitamos que las consultas al servidor sean a estos mismo archivos, de esta forma nos abstraemos de las consultas a la base de datos. Para ello en el constructor de nuestra clase recibimos un parámetro de tipo *String* que representa la dirección a la que realizar la consulta y otro parámetro de tipo *String* que representa el comando a enviar. La implementación pasa a ser en tecnología Java a partir de aquí ya que usaremos los métodos auxiliares para comunicación que nos provee. Hemos de aclarar que el método *onPostExecute()* no alberga funcionalidad alguna, ya que esta tarea asíncrona está destinada sólo al envío de comandos. Sin embargo es el lugar para colocar el control de errores de la comunicación, por ello, cada vez que creemos un objeto de este tipo, deberemos sobrescribir este método para tal fin. Una vez creado un objeto *AsyncTask* lo ejecutaremos llamando al método **execute()**.

Con todo lo escrito anteriormente creemos que queda explicada la implementación de la aplicación Android ya que esta tecnología está ampliamente extendida en estos días y no es tan desconocida como para entrar en más detalle. Hemos de mencionar que hemos creado otro tipo de tarea asíncrona para la obtención de datos, de manera casi similar a la descrita anteriormente, y que la aplicación ha sido implementada de forma que podamos extenderla fácilmente. Cada fragmento contiene una lista de componentes que con sólo añadir un elemento más la aplicación, ésta se encarga de generar la interfaz gráfica para su control.

Para finalizar este apartado, recordamos que en el DVD que entregamos con todo el material del proyecto se incluyen vídeos de demostración en el que podemos ver, entre otras cosas, la aplicación Android ejecutándose.

7 Manual de usuario

A continuación incluimos un manual de usuario para describir qué acciones podemos llevar a cabo en nuestro sistema domótico. Antes de continuar, nos gustaría recordar que tenemos como uno de los requisitos del proyecto que las interfaces de control sean **amigables** y/o que requieran **poco tiempo** para aprender a hacer un buen uso de ellas. Esto es algo que hemos tenido presente a la hora de implementarlas y que esperamos se vea reflejado en este manual.

7.1 Aplicación web

En primer lugar, comenzamos con la aplicación web. Lo primero que veremos es la ventana de *Login* que nos da acceso a las páginas que contienen los controladores (*figura 7.1*). Como hemos descrito en el capítulo de implementación su organización está basada en el patrón *Maestro-Detalle* en el que el Maestro representa la estructura de la maqueta y el Detalle contiene todos los componentes que podremos manejar en cada habitación (ver *figura 7.2*).

Para acceder, por ejemplo, a las luces de la cocina de la maqueta, bastaría con que pulsásemos sobre el botón que se encuentra en el centro de esta sección en el Maestro y esto nos actualizaría los controles en el Detalle, reflejando los correspondientes a la nueva habitación. Encender un led es un proceso tan **sencillo** como pulsar el botón correspondiente en el Detalle.

Figura 7.1: Ventana de Login.

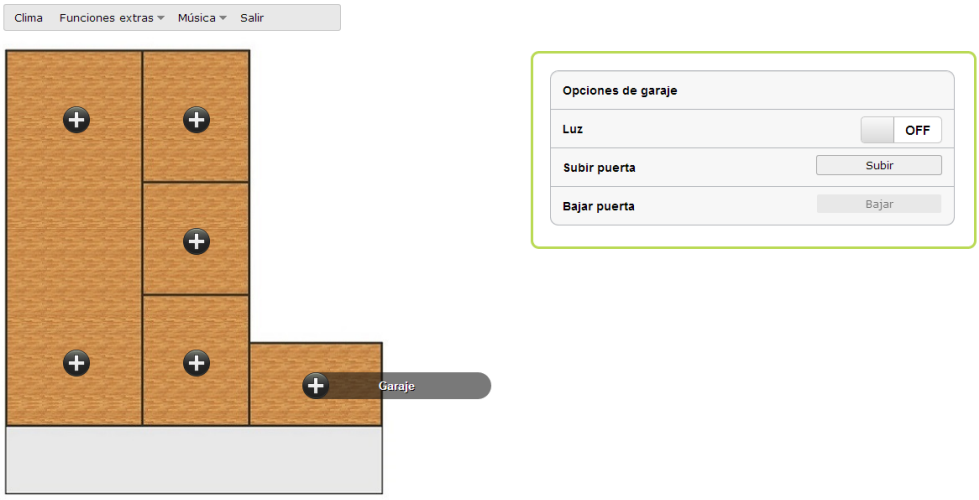


Figura 7.2: Página principal de la web.

La aplicación web no sólo se limita a activar mecanismos o encender luces. También tiene como misión mostrar el estado actual de los sensores e incluso mostrar avisos en casos excepcionales como puede ser la detección de una concentración preocupante de gases en la cocina. Si nos fijamos en la figura 7.3 podemos ver un ejemplo de motorización de la maqueta. Desde la pestaña *Clima* del menú de funciones podemos mostrar una gráfica en la que hemos recogido los datos referentes a la temperatura y a la humedad de nuestra maqueta a lo largo del día, así como los datos actuales.

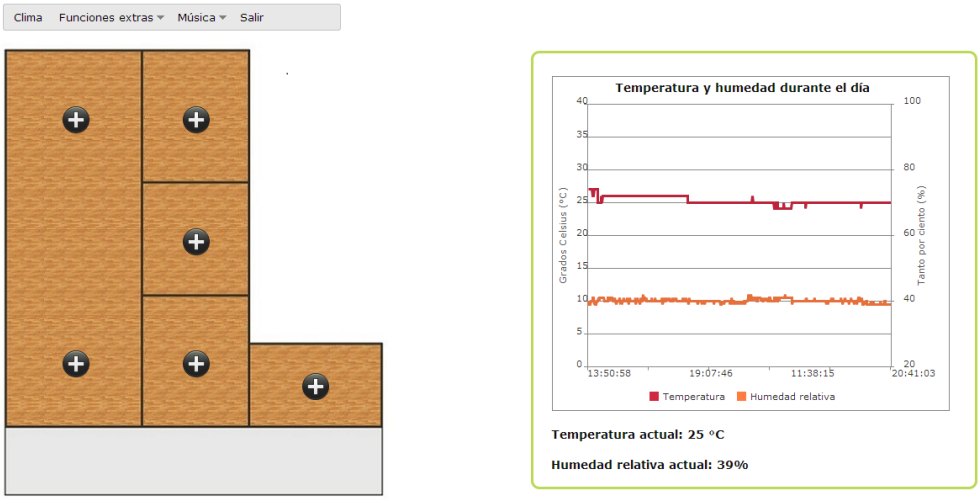


Figura 7.3: Pestaña de clima.

También, en este anteriormente mencionado menú de funciones encontramos otras pestañas desde las que podremos acceder a funcionalidades extras, un servicio de música en *streaming* y hacer *Logout*. Como funcionalidades extras encontramos encender y apagar todas las luces de la casa para una mayor comodidad del usuario. Si pulsamos sobre la pestaña de *Música* podremos reproducir una canción o subir otra al servidor para cuando queramos escucharla. Podemos ver las figuras 7.4, 7.5 y 7.6 para ilustrar lo anteriormente mencionado.

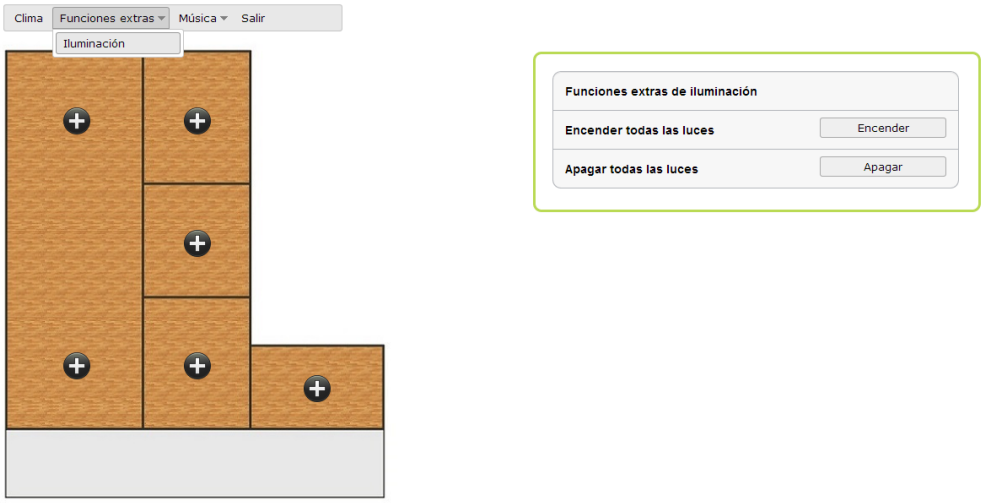


Figura 7.4: Funciones extras: iluminación.

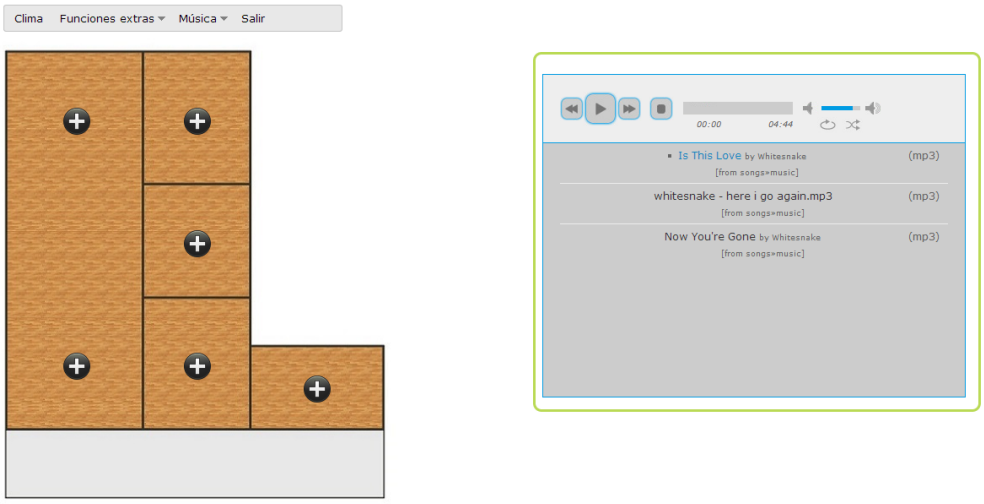


Figura 7.5: Servicio de reproducción de música en streaming.

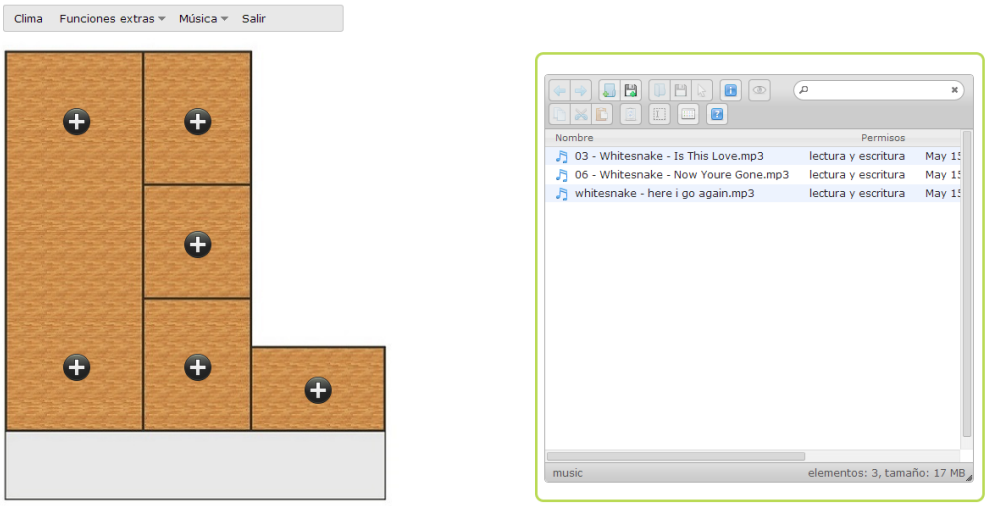


Figura 7.6: Servicio de subida de archivos de audio.

Por último, referente a esta aplicación de control web, contamos con avisos en forma de alertas en casos especiales. Por ejemplo, si se dispara el sensor de gas de la cocina, se activará una alarma y nos aparecerá una alerta en la web. La única forma de apagar esta alarma es cerrando el aviso en la aplicación lo que significará que somos conscientes del riesgo.

7.2 Aplicación Android

A continuación, describiremos la segunda aplicación de control del sistema, la aplicación de nombre "*DomóticaApp*"desarrollada en la tecnología *Android*.

Partimos de las ideas mencionadas anteriormente: sencillez y eficacia. La interfaz debe ser amigable y potente a la vez. A cualquier usuario habitual de la tecnología Android le resultará extremadamente sencillo el manejo de esta aplicación.

En primer lugar nos encontramos con la ventana de *login* como puede verse en la *figura 7.7* (hemos decidido usar el término ventana para referirnos a cada una de las distintas partes de la interfaz de usuario de la aplicación). Esta ventana nos requerirá unos credenciales de usuario para permitirnos el acceso al control del sistema y, a su vez, nos mantendrá informados de la validez de los datos introducidos.

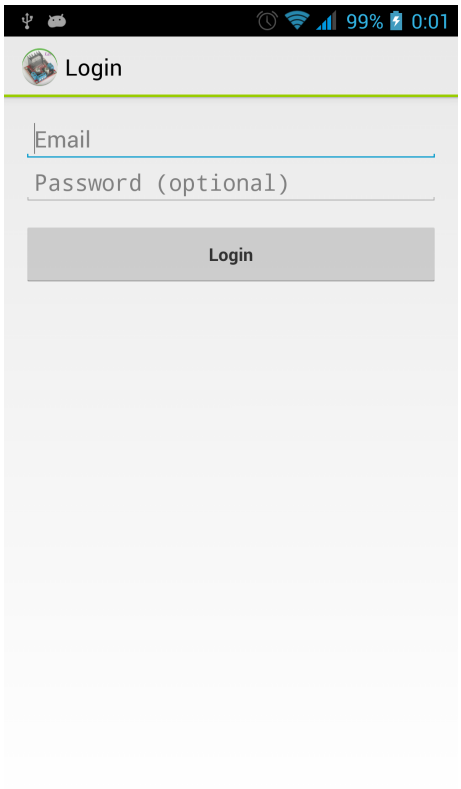


Figura 7.7: Ventana de login.

Una vez que hayamos accedido al sistema podremos ver una ventana como la mostrada en la *figura 7.8*. Desde aquí tendremos el control absoluto del sistema. Como puede verse en la figura anteriormente mencionada disponemos de unas pestañas de navegación, cada una representa a una zona de la casa. Si queremos acceder a la pestaña que contiene los elementos del garaje bastaría con hacer un movimiento de tocar y deslizar la pantalla hasta llegar a ella, o bien desplazando las pestañas hasta que la deseada sea visible y posteriormente pulsando en ella.

Los controles son muy simples, contamos con botones de encendido y apagado para luces, botones de acciones para las puertas, botones de progresión como el mostrado en la *figura 7.9* para luces regulables y simples campos de texto para información acerca del estado del sistema (temperatura y humedad). Cada botón está diseñado para ofrecer el estado de los componentes, de manera que si pulsamos en uno, sabemos en que estado se encuentra el elemento de la casa que sufra la funcionalidad de ese botón.

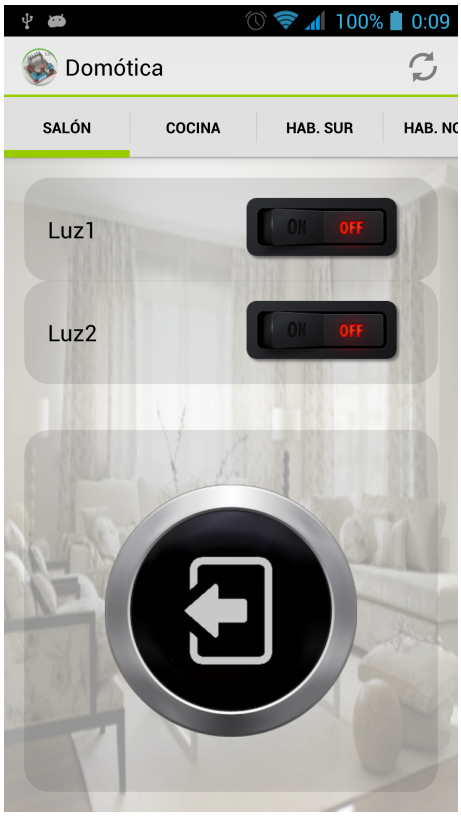


Figura 7.8: Ventana principal de la aplicación.

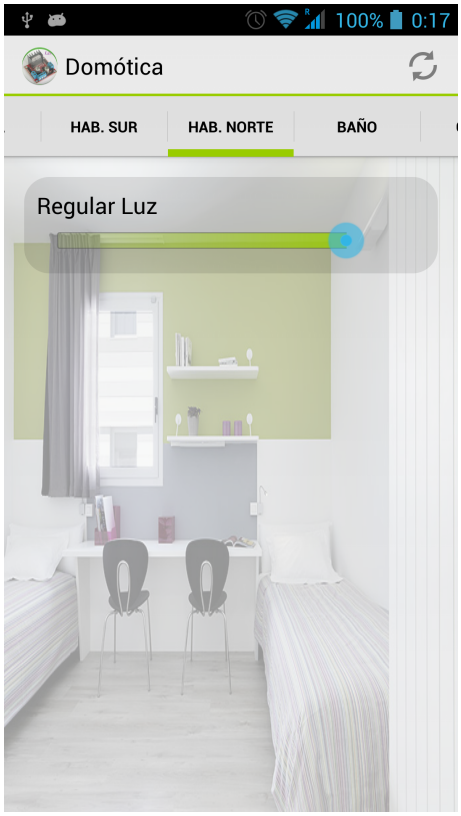


Figura 7.9: Botón de progreso.

La aplicación debe mantener en coherencia el estado de los botones respecto al estado

de los componentes físicos del sistema. Por tanto la aplicación cuenta con la funcionalidad de refresco de la interfaz de usuario. Este refresco podemos ejecutarlo de dos formas: al inicio de la aplicación o activado manualmente por el usuario. La primera de las opciones está desactivada por defecto y tendríamos que activarla si quisiésemos que el refresco se realizase de esta forma. La segunda opción es que pulsásemos el botón de la esquina superior derecha de la aplicación de manera que refrescaríamos los controles de la ventana actual.

Por último, las alertas del sistema se muestran en cuadros de dialogo como el mostrado en la *figura 7.10*. Si alguna cosa ocurriese en la casa, nos llegaría la notificación al instante a nuestra aplicación. Además, podremos desactivar la alarma de la casa pulsando el botón que aparece en la parte inferior del cuadro de diálogo.

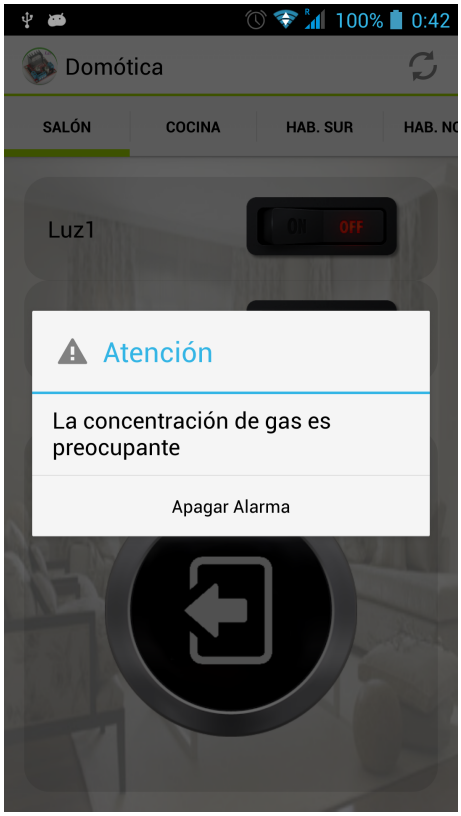


Figura 7.10: Cuadro de diálogo de alerta.

8 Pruebas

Hemos decidido hacer tres tipos de pruebas en este proyecto:

- **Pruebas de concepto**: Nos referimos a aquellas pruebas cuya finalidad es probar la viabilidad de un diseño y su posterior implementación.
- **Pruebas unitarias**: Aquellas pruebas cuyo objetivo es probar componentes del sistema aisladamente.
- **Pruebas de integración**: Pruebas finales, dedicadas a comprobar el correcto funcionamiento del sistema completo, o partes que engloban varios componentes.

8.1 Pruebas de concepto

8.1.1 Circuito simple con Arduino

Esta prueba de concepto surge de la necesidad de comprobar el buen estado de los componentes electrónicos usados en el proyecto, así como refrescar nuestros conceptos de electrónica.

Componentes utilizados

Los componentes que hemos usado en esta prueba son los siguientes:

- Microcontrolador Arduino Mega 2560.
- Cables de conexión.
- Placa "BreadBoard".
- Diodo LED de alta luminosidad (20 Kmcd).
- Resistencia (470 Ω).
- PC con el entorno de desarrollo de Arduino instalado.

Desarrollo de la prueba

Para esta prueba simple comenzamos montando el **circuito** de la figura 8.1. La clave de este montaje reside en emplear una resistencia adecuada al diodo LED que colocaremos. Aunque Arduino ya provee de una resistencia interna, ésta está desactivada por defecto por lo que hace uso de una externa. Para el cálculo de la resistencia necesaria, debemos tener la intensidad de corriente que tiene que atravesar el LED, ya que ésta no es proporcionada por el fabricante hicimos uso de información suministrada a través de Internet. Concluimos que para un LED con caída de potencial entre sus extremos de 2 a 3 V, necesitamos una resistencia de unos 470 Ω.

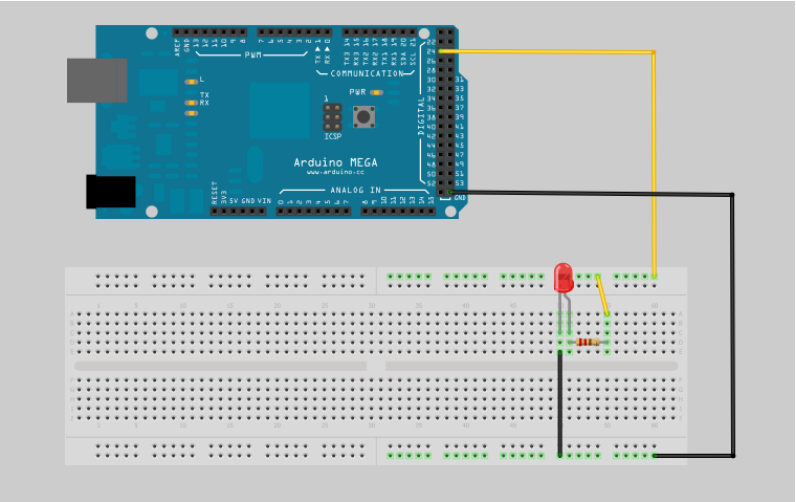


Figura 8.1: Vista simulada del circuito.

Una vez que sabemos este dato, el resto del montaje es sencillo. Para alimentar el circuito hacemos uso de los **pines digitales** del microcontrolador, que proporcionan una salida de 0 V en su estado "LOW" y +5 V en su estado "HIGH".

Software

Pasamos ahora a la parte software de la prueba que viene dada por el siguiente código en lenguaje Arduino y sus correspondientes comentarios.

Listing 8.1: circuitoSimple.ino

```
// declaramos una variable led que representará al pin del microcontrolador usado
int led = 24;


void setup() {
  // inicializamos el pin como salida
  pinMode(led,OUTPUT);
}


void loop() {
  //en el bucle, encendemos el LED poniendo el pin al que está conectado en HIGH (+5V)
  digitalWrite(led,HIGH);

  //dejamos 1000 milisegundos el led encendido
  delay(1000);

  // y lo apagamos
  digitalWrite(fan,LOW);
  delay(1000);
}
```

Una vez cargado el programa en el microcontrolador observamos como la prueba tiene éxito. El diodo LED se enciende y se apaga con un segundo de retardo en cada operación.

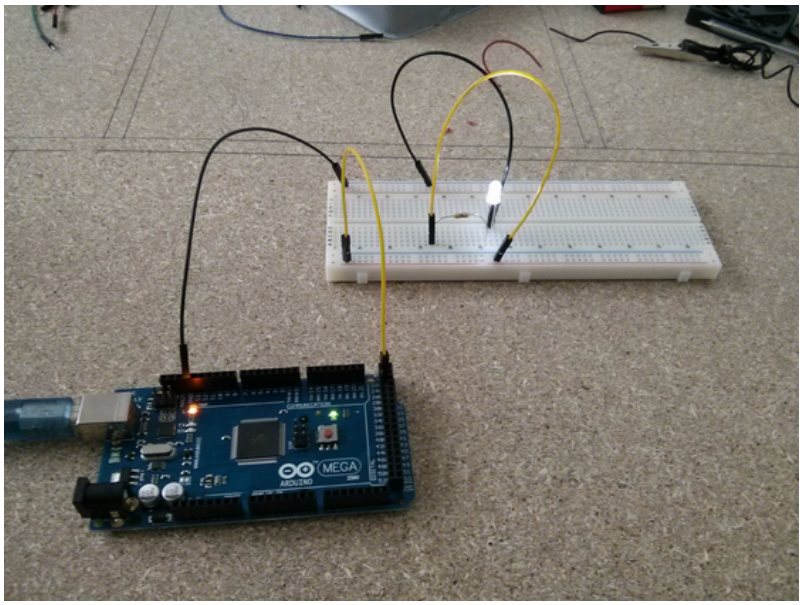


Figura 8.2: Circuito de led.

8.1.2 Comunicación entre Arduino y Raspberry Pi

El objetivo de esta prueba es la comunicación entre los dispositivos Arduino y Raspberry Pi, por lo que la prueba se divide en dos partes:

- Envío de datos desde Arduino a Raspberry Pi.
- Envío de datos desde Raspberry Pi a Arduino.

Antes de todo, debemos elegir el **medio** por el que Arduino y Raspberry Pi se van a comunicar . Para ello, hemos elegido la opción más sencilla y segura que consiste en conectar estos dispositivos mediante un **cable USB** (Tipo-A Tipo-B) conectado a sus respectivos puertos USB. De este modo, Arduino se sirve de la Raspberry Pi para alimentarse y poder funcionar por lo que no nos debemos preocupar por conseguir un cargador para esta prueba. Aquí vemos cómo queda la conexión:

Tras tener claro el medio de comunicación, configuramos Raspberry Pi para que a través del lenguaje **Python** podamos enviar y recibir datos a Arduino. Para que esto sea posible, instalamos la librería *python-serial*, encargada de control de puertos serie en Python, con el siguiente comando:

- *sudo apt-get install python-serial*

Con esto ya tenemos lista la Raspberry Pi para comunicarse con el puerto serie de Arduino desde Python. Ahora sólo nos queda determinar en qué puerto está conectado nuestro Arduino. Para ello, con el siguiente comando podemos ver la lista de puertos y comprobar qué puerto es el asignado:

- *ls /dev/tty**

En nuestro caso el puerto es /dev/ttyACM0, por lo que ya podemos empezar la prueba.

De Arduino a Raspberry Pi

En esta primera parte vamos a enviar el mensaje "Hola, mundo" desde Arduino a Raspberry Pi. Debemos prestar atención a los métodos *Serial.println()*, de Arduino y *read()* de Raspberry Pi que son los que llevan a cabo las tarea de enviar y recibir datos respectivamente.

1. Primero debemos cargar el código, indicado a continuación, a Arduino:

Listing 8.2: Arduino2RPi.ino

```
void setup() {
  //Establecemos la velocidad de transmisión de datos del puerto serie
  Serial.begin(9600);
}

void loop() {
  //Enviamos el mensaje cada dos segundos por el puerto serie
  Serial.println("Hola Mundo");
  delay(2000);
}
```

2. Después, debemos crear un archivo Python en la Raspberry Pi con el siguiente código:

Listing 8.3: prueba.py

```
#Importamos la librería python serial para comunicarnos con el puerto serial
import serial
#Indicamos el puerto serial correspondiente a Arduino y la velocidad de trasnmisión
ser=serial.Serial('/dev/ttyACM0',9600)

while 1:
    #Mostramos por pantalla los datos que recibimos de Arduino
    print(ser.read());
```

3. Conectamos los dispositivos entre sí y ejecutamos el archivo Python. Si todo ha salido bien la Raspberry Pi muestra el mensaje siguiente sucesivamente:

```
>>> ========== RESTART ==========
>>>
Hola, mundo
>>>
```

Con esto podemos obtener datos de Arduino como por ejemplo la temperatura.

De Raspberry Pi a Arduino

Este caso es muy similar al anterior. Debemos prestar especial atención a los métodos *Serial.read()*, para leer los datos enviados desde Raspberry Pi a Arduino, y *write()*, para enviar datos a Arduino desde Raspberry Pi en Python y, por último, destacamos que debemos establecer un pequeño tiempo de espera mediante el método *time.sleep()* en el archivo Python para que Arduino pueda recoger en su *buffer* los datos recibidos.

1. Cargamos el código siguiente a Arduino:

Listing 8.4: RPI_2_Arduino.ino

```
char response[20]="";//Array para almacenar el mensaje recibido
int index=0;//Índice para añadir los caracteres al array response

void setup() {
 //Establecemos la velocidad de transmisión del puerto serie
  Serial.begin(9600);
}

void loop() {
  //Mientras que haya datos en el buffer los guardamos en el array response
  while(Serial.available()){
    response[index++]=Serial.read();
  }
  //Mostramos el mensaje que recibimos de Raspberry Pi
  Serial.println(response);
  index=0;
  delay(2000);
}
```

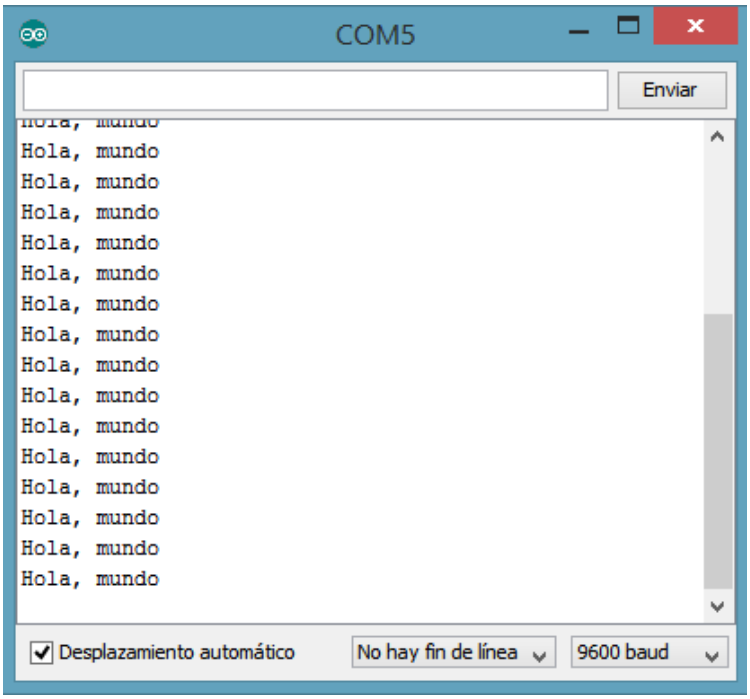
2. A continuación, en un archivo Python alojado en la Raspberry Pi introducimos el siguiente código:

Listing 8.5: prueba.py

```
#Importamos la librería python serial para comunicarnos con el puerto serial
import serial
#Importamos la librería time para establecer un tiempo de espera
import time
#Indicamos el puerto serial correspondiente a Arduino y la velocidad de trasnmisión
ser=serial.Serial('/dev/ttyACM0',9600)

while 1:
    #Enviamos un mensaje a Arduino cada segundo
    ser.write('Hola, mundo');
    time.sleep(1)
```

3. Por último, conectamos los dispositivos y ejecutamos el archivo Python. Si el proceso se ha realizado con éxito, deberíamos ver, desde el entorno de desarrollo de Arduino, lo que éste ha recibido:



De esta forma podemos enviar ordenes a Arduino para que haga una acción concreta, por ejemplo, encender un led cuando reciba el carácter 'H'. Por tanto, dependiendo del mensaje recibido Arduino actuará de una forma u otra dándonos más flexibilidad en el control del dispositivo.

8.1.3 Control de Arduino desde aplicación web - Método 1

En esta prueba pretendemos controlar Arduino mediante una aplicación web. Para ello, Raspberry Pi será el servidor web donde se alojan los archivos necesarios con el objetivo de enviar órdenes, en este caso de apagado y encendido, a Arduino.

Preparación de Arduino

Hemos diseñado un circuito simple compuesto por un led conectado al pin 46 y una resistencia de 470 Ω como muestra la figura:

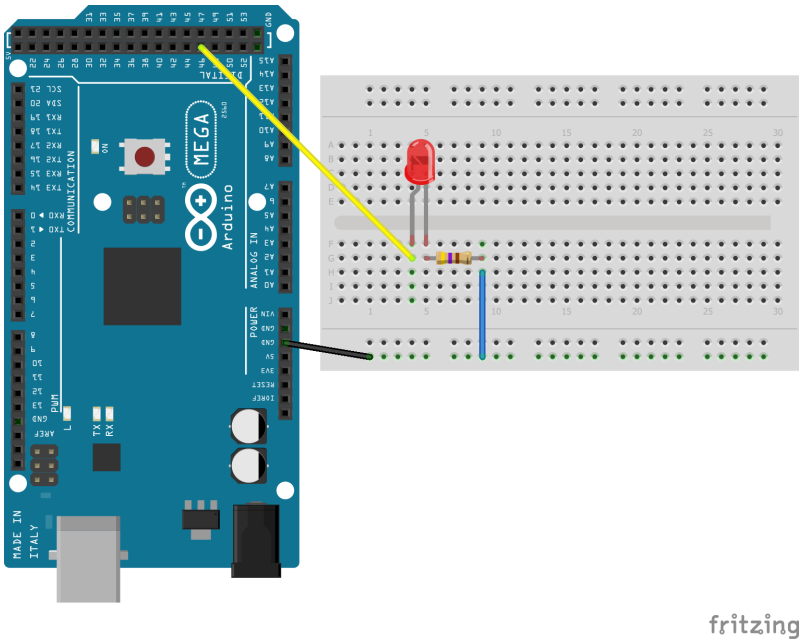


Figura 8.3: Circuito de led.

Además, hemos desarrollado un sencillo programa para Arduino que interpreta los comandos recibidos desde la Raspberry Pi.

Configuración de Raspberry Pi

Para preparar la Raspberry Pi para esta tarea debemos instalar la librería *python-serial* que se hace cargo de la comunicación entre Raspberry y Arduino y la librería web.py para poder desarrollar páginas web en Python.
Las librerías *python-serial* y *web.py* las instalamos mediante estos comandos:

- *sudo apt-get install python-serial*
- *sudo apt-get install python-webpy*

Configuración del servidor

Procedemos a instalar en la Raspberry Pi el servidor Apache y el lenguaje PHP para el desarrollo de webs dinámicas con el siguiente comando:

- sudo apt-get install apache2 php5 libapache2-mod-php5

Posteriormente, reiniciamos el servicio:

- sudo service apache2 restart

Comprobamos la conexión con el servidor introduciendo la IP de Raspberry Pi en el navegador. Si el proceso ha sido realizado con éxito saldrá el siguiente mensaje:

Diseño de la página web

Todo lo referente al servidor web lo guardaremos en el directorio `/var/www` de Raspberry Pi. Dicho directorio contiene dos carpetas que utilizará la librería *web.py*:

- `static`: contiene archivos como hojas de estilo CSS, JavaScript, imágenes, etc.
- `templates`: contienen los archivos HTML pertenecientes a la página web.

El resto de archivos los almacenaremos en el directorio en `/var/www`. La página web llamada *prueba.html* se compone de un campo de entrada de texto HTML en el que introduciremos las letras '*H*' o '*L*', ya sea para encender o apagar el led, y un botón Enviar con el que transmitiremos el texto introducido en el campo comentado anteriormente.

Para poder poner en funcionamiento la página web, primero creamos un archivo en Python, *app.py*, en el directorio `/var/www`, con el que vamos a ejecutar el servidor web. La función de *app.py* es recibir los datos introducidos en la página web ('*L*' y '*H*') para enviárselos a Arduino. En dicho archivo detallamos lo siguiente:

- Importación de las librerías necesarias.
- Establecimiento de la comunicación con Arduino.
- *web.py* define la relación entre *prueba.html* con la clase *prueba*.
- Métodos GET, con el que solicitamos la página web, y PUT, con el que enviamos datos al servidor.
- Ejecución *web.py*.

Una vez que terminamos el *script* en Python, procedemos a ejecutarlo. Si está todo listo el servidor devolverá la dirección `http://0.0.0.0:8080/`. Es entonces ahora cuando podemos acceder a la página web desde la dirección IP de Raspberry Pi y el puerto 8080 (192.168.1.138:8080). Si introducimos los caracteres '*H*' o '*L*' el led debería responder a nuestra solicitud encendiéndose o apagándose.

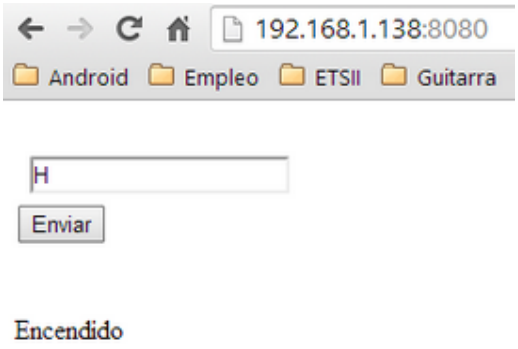


Figura 8.4: Resultado de la página web.

Código desarrollado para la prueba

A continuación presentamos los principales archivos que hacen posible la ejecución de la prueba con éxito:

Listing 8.6: app.py

```
import web #Importamos la librería web para el funcionoamiento del servidor web
import serial #La librería serial la necesitamos para comunicarnos con Arduino

urls = ('/','prueba')#Indicamos qué nombre tiene nuestra página web
#Indicamos donde se sitúa almacenada nuestra página web
render= web.template.render('/var/www/templates')
#Establecemos el puerto y la velocidad de transmisión de nuestro Arduino
arduino=serial.Serial('/dev/ttyACM0',9600)
app=web.application(urls,globals())#web.py mapea la web con la clase de su mismo nombre

my_form=web.form.Form(web.form.Textbox('',class_='textfield', id='textfield'),)
#Definimos una clase con el mismo nombre del archivo HTML para que web.py pueda
#asociar prueba.html con la clase prueba.
class prueba:
    #Funciones GET y POST. En cada una de ellas obtenemos una copia del formulario
    #mediante my_form()
    def GET(self):
        form=my_form()
        #Devuelve el archivo prueba.html pasándole el formulario  y un texto deseado
        return render.prueba(form,"Eventos")
    def POST(self):
        form = my_form()
        form.validates()
        #Se obtiene el valor que hemos introducido en el campo en el campo input
        #del formulario
        comando= form.value['textfield']
        #Si el valor que introducimos es una 'H', se envía dicho valor a Arduino
        #y éste responde encendiendo un led. Por el contrario, se apaga si enviamos
        # la letra 'L'.
        if comando =='H':
            arduino.write(comando)
            return 'Encendido'
        if comando =='L':
            arduino.write(comando)
            return 'Apagado'
        else:
            return 'Inserte comando valido'
#Ejecutamos web.py
if __name__ == '__main__':
    app.run()
```

Listing 8.7: PruebaWeb.ino

```
//Declaramos el numero de pin para el led que vamos a controlar
const int led = 46;

void setup () {
  //Declaramos el pin led=7 como salida
  pinMode(led, OUTPUT);
  //Inicializamos el puerto serial a 9600 baudios
  Serial.begin(9600);
}

void loop () {
  //Si hay datos en el buffer de recepción
```

```
  if (Serial.available()) {
    //Guardamos en c cada byte del buffer
    char c = Serial.read();
    //Mostramos c
    Serial.println(c);
    //Si es H (HIGH) encendemos el led
    if (c == 'H') {
      digitalWrite(led, HIGH);
    }
    //Si es L (LOW) apagamos el led
    else if (c == 'L') {
      digitalWrite(led, LOW);
    }
  }
}
```

8.1.4 Control de Arduino desde aplicación web - Método 2

Con esta prueba pretendemos controlar Arduino mediante una aplicación web. A diferencia del método 1, con este nuevo procedimiento nos deshacemos del uso de la librería *web.py* que nos ayuda a implementar los métodos GET y POST del servidor web, así como el arranque del mismo y el renderizado de la página web. Como novedad, utilizaremos PHP 5 que nos hará las cosas mucho más fáciles.

Seguimos con la misma dinámica: usamos la Raspberry Pi como servidor web con el objetivo de enviar órdenes a Arduino para encender y apagar un led a través de dos archivos en Python que envían las órdenes de encendido y apagado a Arduino.

Preparación de Arduino

Nos apoyamos en el circuito que ya diseñamos en el método 1 compuesto por un led conectado al pin digital 46 de Arduino y una resistencia de 470 ohmios.

Además, cargamos en Arduino el mismo programa que enciende o apaga el led si recibe los caracteres 'H' o 'L' visto en el método 1.

Configuración de Raspberry Pi

A continuación, instalamos la librería *pySerial* encargada de la comunicación serie entre Arduino y Raspberry Pi:

- *sudo apt-get install python-serial*

Configuración del servidor web

Debemos instalar un servidor web, en nuestro caso hemos elegido Apache 2, y el módulo correspondiente para su integración con PHP 5 en Raspberry Pi. Para ello usaremos el mismo comando ya visto en el anterior método y al finalizar reiniciamos el dispositivo:

- *apt-get install apache2 php5 libapache2-mod-php5*

Diseño de la página web

Los archivos necesarios para el funcionamiento de esta prueba, exceptuando el código de Arduino, los almacenaremos en la carpeta `/var/www` de Raspberry Pi, la cual pertenece al servidor web. Al no usar *web.py* no hace falta estructurar el contenido de esa carpeta, por

tanto, podemos copiar todos los archivos necesarios directamente en ella. Dichos archivos son:

- index.html
- encender.php
- apagar.php
- enciendeled.py
- apagaled.py

La página web (*index.html*), basada en HTML5, consta de dos formularios: un formulario para encender el led y otro para apagarlo. Cada formulario contiene un botón que al pulsarlo ejecutará el correspondiente archivo PHP que más adelante describimos.

Utilizamos el lenguaje PHP para ejecutar los correspondientes archivos desarrollados en Python. Por tanto, el formulario necesario para encender el led ejecutará el archivo *encender.php*, y éste mediante la instrucción *exec()* ejecutará el archivo en Python, *enciendeled.py*, encargado de enviar la orden a Arduino. Del mismo modo hacemos para apagar el led.

Como conclusión, podemos añadir que con este método la tarea de programar se vuelve más sencilla y, además se crea archivos dedicados a la ejecución de una determinada orden haciendo la tarea de detectar fallos más rápida. La parte negativa es que debemos esperar un tiempo bastante considerable (2 segundos) para la comunicación entre dispositivos.

Resolución de problemas experimentados durante la prueba

Para evitar problemas de sincronización entre la Raspberry Pi y Arduino al comunicarse, hemos establecido un tiempo de espera en los archivos Python.

Por último, al probar la página web y pulsar sobre uno de los botones hemos experimentado problemas a la hora de poder ejecutar el archivo Python correspondiente a la acción que deseábamos realizar. Ésto es debido a que el usuario que accede a la web desde su dispositivo no tiene permisos para utilizar la instrucción *exec()* de PHP encargada de la ejecución del archivo Python. Para darle permisos a cualquier usuario que acceda desde Internet a nuestra web debemos realizar los siguientes pasos sobre Raspberry Pi:

1. Abrimos el archivo *000-default* que corresponde al servidor web con el siguiente comando:
   - *sudo nano /etc/apache2/sites-enabled/000-default*
2. En el archivo buscamos la sección <Directory /var/www/> y cambiamos la línea `-AllowOverride None-` por `-AllowOverride All-` y guardamos el archivo.
3. Añadimos el usuario de Apache, *www-data*, a la lista de usuarios con permisos de administrador:

- *sudo visudo*

4. Al final del archivo añadimos la siguiente línea y lo guardamos:
   - www-data ALL=(ALL) NOPASSWD: ALL
5. Por último, reiniciamos el servidor Apache:
   - `sudo /etc/init.d/apache2 restart`

Código desarrollado para la prueba

A continuación, presentamos algunos de los archivos más relevantes de la prueba:

Listing 8.8: encender.ino

```
<?php
//Ejecutamos el archivo encender.py para encender el led
$apaga = exec('sudo python /var/www/raspduino/enciendeled.py');
//Volvemos a la página principal
header('Location:index.html');
?>
```

Listing 8.9: enciendeled.py

```
import serial #Librería para la comunicacion del puerto serial
import time #Librería para establecer tiempo de espera
#Etablecemos el puerto conectado a Arduino y su velocidad de trasmisión
arduino = serial.Serial('/dev/ttyACM0',9600)
time.sleep(2)#Dejamos un tiempo de espera para preparar la conexión
arduino.open()#Abrimos conexión
comando='H'
arduino.write(comando)#Enviamos 'H' a Arduino para encender el led
arduino.close()#Cerramos la conexión
```

8.1.5 Control de Arduino desde aplicación web - Método 3

Con esta prueba pretendemos controlar Arduino mediante una página web. La diferencia principal con el resto de métodos es que no usamos el lenguaje Python en ningún momento puesto que ahora la comunicación con Arduino la realizamos a través del lenguaje PHP.

Como hemos hecho en los anteriores métodos, el servidor web del que obtendremos la página web seguirá estando alojado en Raspberry Pi y dicho dispositivo enviará a Arduino las órdenes 'H' o 'L' para encender o apagar un led.

Preparación de Arduino

Diseñamos un sencillo circuito formado por una resistencia de 470 ohmios y un led conectado al pin digital 46 de Arduino.

Cargamos en Arduino el programa necesario para encender o apagar el led si recibe los caracteres 'H' o 'L', usado ya en los anteriores métodos.

Configuración de Raspberry Pi

A continuación, instalamos la librería *pySerial* encargada de la comunicación serie entre Arduino y Raspberry Pi:

- *sudo apt-get install python-serial*

Configuración del servidor web

Como ya hemos visto en ocasiones anteriores, necesitamos instalar un servidor web y el módulo que le permita integrarse con PHP en Raspberry Pi. Vamos a instalar Apache 2 y para ello usaremos el mismo comando ya visto en los demás métodos y al finalizar reiniciamos el dispositivo:

- *apt-get install apache2 php5 libapache2-mod-php5*

Diseño de la página web

A continuación, creamos los archivos que necesitamos almacenar en el directorio del servidor web Apache `/var/www` para el control desde la página web:

- index.html
- arduino.php

La página web, *index.html*, consta principalmente de dos formularios: el primero para encender el led y el segundo para apagarlo. Cada formulario tiene un botón tipo *submit* y un campo de entrada de texto, *input*, que para el usuario no es visible y cuyo valor es el comando que deseamos enviar a Arduino. Luego si deseamos encender el led, simplemente pulsamos sobre el botón de encendido, desencadenando el envío del carácter 'H' que es el valor del campo de entrada de texto correspondiente.

Con el archivo PHP, *arduino.php*, obtenemos el carácter que enviamos desde la página web, ya sea 'H' o 'L'. Posteriormente, debemos establecer la comunicación con Arduino y enviarle el carácter. Para ello, hemos decidido usar la clase *php-serial*, un archivo en lenguaje PHP desarrollado por Rémy Sánchez, que incluiremos en nuestro archivo *arduino.php*. Llamando a instrucciones de esta clase estableceremos la conexión de una manera rápida y sencilla. Si todo está correcto podemos encender y apagar el led desde la página web.

Para concluir, podemos decir que con éste método agilizamos la tarea de programar y eliminamos el retraso en el establecimiento de la comunicación entre Raspberry Pi y Arduino que necesitamos en los otros métodos.

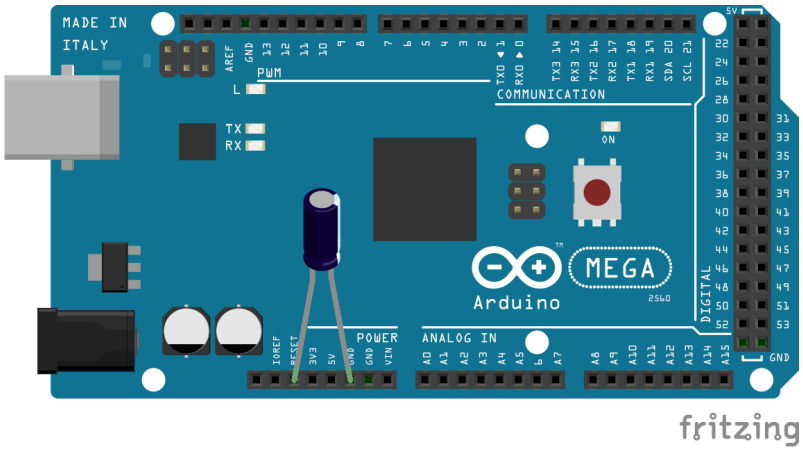
Resolución de problemas experimentados durante la prueba

El primer problema que ha surgido es la imposibilidad de establecer la comunicación con Arduino mediante PHP si lo hacíamos desde la página web. Dado que Raspberry Pi y Arduino están conectados entre sí mediante un cable USB (TIPO A - TIPO B), para comunicarse entre ellos, necesitamos establecer una comunicación serie. Entonces, cuando pulsamos sobre cualquier botón para accionar el led estamos estableciendo esa comunicación pero como usuario del servidor web Apache, *www-data*, el cual no tiene suficientes permisos para controlar el puerto serie.

La solución es bien sencilla, simplemente añadimos dicho usuario al grupo *dialout* de Raspberry Pi. Todo usuario que se encuentre en dicho grupo tendrá permiso para manejar el puerto serie. Para ello ejecutamos el siguiente comando y ya funciona nuestra página web:

- *sudo adduser www-data dialout*

El segundo problema ocurre cuando Arduino recibe el comando. Por defecto, el puerto serie de Arduino desactiva la línea DTR y activa el pin RESET cada vez que termina de recibir datos, por tanto, cuando enviamos los comandos, 'H' o 'L', cerramos la conexión y Arduino se reinicia perdiendo el comando que acaba de recibir. Ésto es una comportamiento normal en Arduino y podemos evitarlo colocando un condensador de 10μF ó 100μF entre los pines GND y RESET como explica la web oficial de Arduino.



Código desarrollado para la prueba

El código más importante de la prueba es el desarrollado en PHP que mostramos a continuación:

Listing 8.10: arduino.ino

```
<?php
//Incluímos la librería necesaria para comunicarnos con el puerto serie mediante php
include ('php_serial.class.php');
//Comprobamos que lo que recibimos por el método POST no sea nulo
if( isset($_POST['componente']) ){
        //Guardamos lo recibido en la variable comando
        $comando=$_POST['componente'];
        //Creamos una instancia de phpSerial para establecer la conexión
        $serial = new phpSerial;
        //Establecemos el puerto de Arduino para la conexión
        $serial->deviceSet("/dev/ttyACM0");
        //Establecemos la velocidad de transmisión de datos
        $serial->confBaudRate(9600);
        //Establecemos el bit de paridad a ninguno
        $serial->confParity("none");
        //Establecemos la longitud de los caracteres enviados a 8 bits cada uno
        $serial->confCharacterLength(8);
        //Establecemos el envío de bits de parada de transmisión
        $serial->confStopBits(1);
        //Abrimos el puerto serie
        $serial->deviceOpen();
        //Enviamos el mensaje
        $serial->sendMessage($comando);
        //Cerramos el puerto serie
        $serial->deviceClose();
        //Volvemos a la página principal
        header('Location:index.html');
}
?>
```

8.2 Pruebas unitarias

8.2.1 Sensor DHT11

Con esta prueba deseamos comprobar el funcionamiento del sensor de humedad y temperatura DHT11. Para ello diseñaremos el circuito entre dicho sensor y Arduino, desarrollaremos el código necesario para obtener los datos recogidos por el sensor y, por ultimo, mostraremos los resultados obtenidos.

Circuito

Mostramos, a continuación, el circuito necesario para la conexión entre Arduino y el sensor DHT11. Debemos prestar atención a la hora de conectar el sensor a Arduino, pues es necesario conectarlo a un **pin analógico** y añadir una **resistencia** de 10 kΩ.

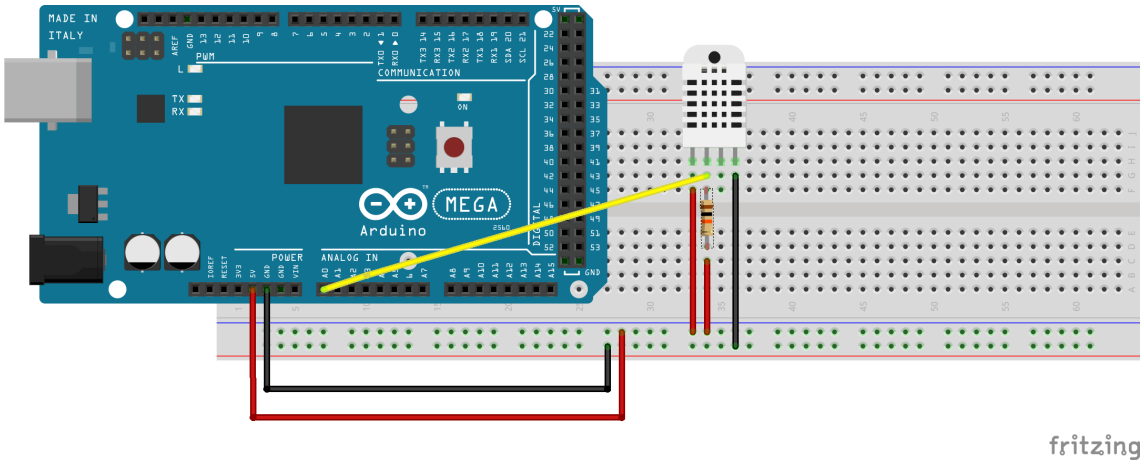


Figura 8.5: Conexión entre Arduino y sensor DHT11.

Código desarrollado para la prueba

Presentamos el código necesario para procesar los valores que obtenemos del sensor DHT11. Destacamos la librería *DHT.h* que nos permite obtener las lecturas del sensor de una manera muy cómoda mediante la función *readHumidity()*.

Listing 8.11: dht11.ino

```
#include <DHT.h>
#define DHTTYPE DHT11
void setup () {
  Serial.begin(9600);
  dht.begin();
}
void loop(){
    int h = dht.readHumidity();
    Serial.print("Humedad relativa: ");
    Serial.println(h);
}
```

Resultados de la prueba

Adjuntamos la imagen con una traza de valores obtenidos por el componente electrónico.

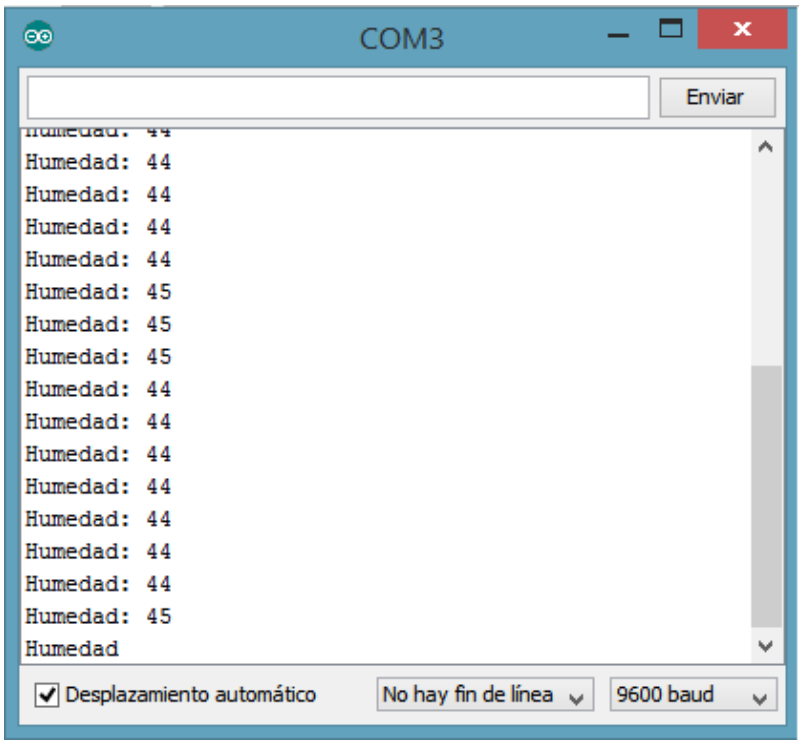


Figura 8.6: Resultados de la prueba del sensor DHT11.

8.2.2 Sensor HC-SR04

Con esta prueba deseamos comprobar el funcionamiento del sensor de ultrasonidos HC-SR04. Para ello diseñaremos el circuito entre dicho sensor y Arduino, desarrollaremos el código necesario para obtener los datos recogidos por el sensor y, por ultimo, mostraremos los resultados obtenidos.

Circuito

Mostramos, a continuación, el circuito necesario para la conexión entre Arduino y el sensor HC-SR04. Debemos conectar los pines que generan y recogen las ondas a pines digitales.

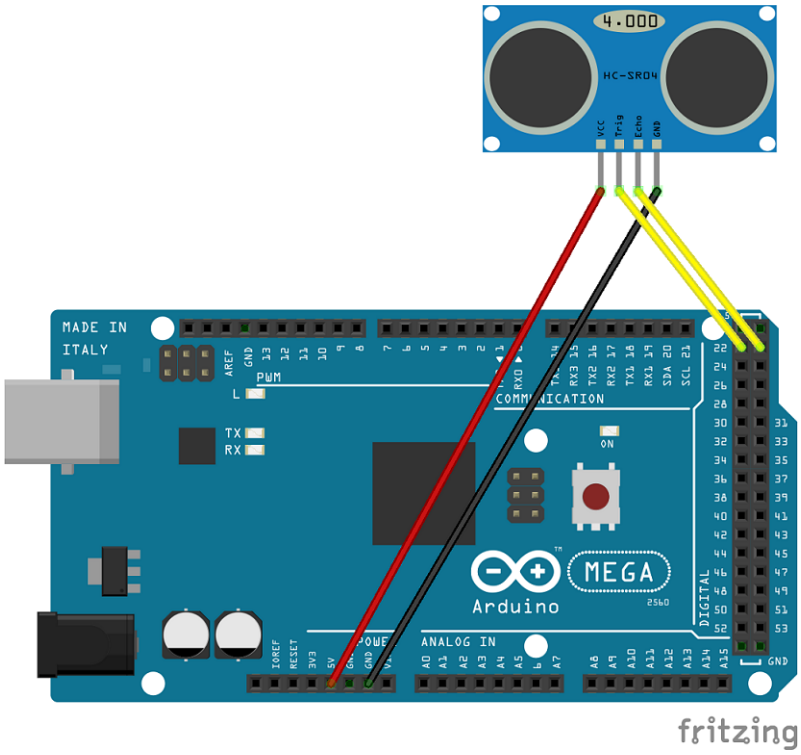


Figura 8.7: Conexión entre Arduino y sensor HC-SR04.

Código desarrollado para la prueba

Presentamos el código necesario para procesar los valores que recogemos del sensor HC-SR04. Hemos utilizado la librería *Ultrasonic.h* que proporciona los mismos resultados que en el caso de obtener el valor del sensor y traducirlo a una unidad de longitud. La librería devuelve la distancia en centímetros entre el sensor HC-SR04 y un objeto que se encuentre en la trayectoria de la onda que emite dicho sensor. La longitud la obtenemos mediante la función *Ranging(CM)*.

Listing 8.12: hcsr04.ino

```
#include <Ultrasonic.h>
Ultrasonic ultrasonico(7,8);
int Pin_echo = 8;
int Pin_trig = 7;
void setup(){
  Serial.begin(9600);
  pinMode(9,OUTPUT);
  pinMode(Pin_trig, OUTPUT);
  pinMode(Pin_echo, INPUT);
}
void loop(){
  Serial.print("Distancia: ");
  int cm= ultrasonico.Ranging(CM);
  Serial.print(cm);
  Serial.println(" cm");
  delay(1000);
}
```

Resultados de la prueba

Adjuntamos la imagen con una traza de valores obtenidos por el componente electrónico ante la presencia de un objeto, en concreto una botella.

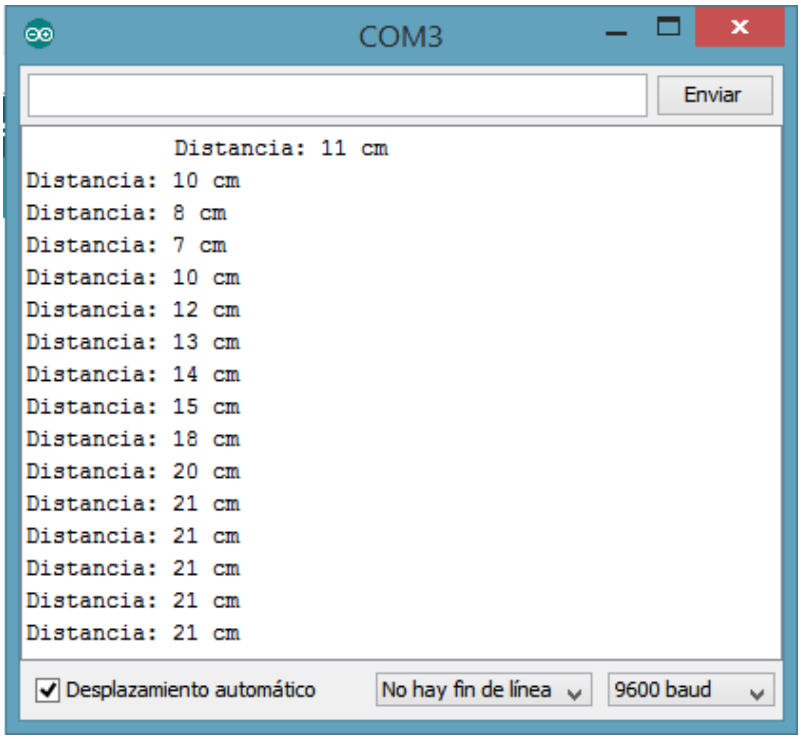


Figura 8.8: Resultados de la prueba del sensor HC-SR04.

8.2.3 Sensor MQ-2

Con esta prueba deseamos comprobar el funcionamiento del sensor de gas MQ-2. Para ello diseñaremos el circuito entre dicho sensor y Arduino, desarrollaremos el código necesario para obtener los datos recogidos por el sensor y, por ultimo, mostraremos los resultados obtenidos.

Circuito

Mostramos, a continuación, el circuito necesario para la conexión entre Arduino y el módulo del sensor de gas MQ-2. Debemos prestar atención a la hora de conectar el sensor a Arduino, pues es necesario conectarlo a un **pin analógico**.

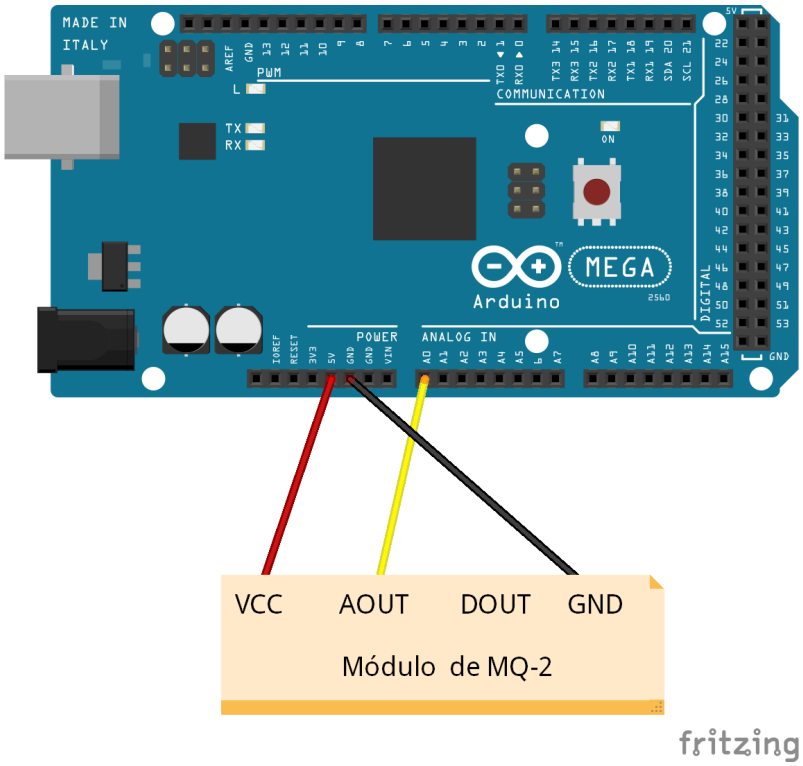


Figura 8.9: Conexión entre Arduino y el módulo del sensor MQ-2.

Código desarrollado para la prueba

Presentamos el código necesario para procesar los valores que obtenemos del sensor MQ-2. Destacamos el proceso para traducir el valor recibido del sensor a la concentración de gas en unidades **ppm** (partes por millón).

Listing 8.13: mq2.ino

```
int mq2Pin=A1;
void setup(){
  Serial.begin(9600);
}
void loop(){
  int sensorValue = analogRead(mq2Pin);
  float vol = sensorValue*(5.0/1023.0);
  int ppm = map(vol,0,5,300,10000);
  Serial.print("Cantidad de gas: ");
  Serial.println(ppm);
}
```

Resultados de la prueba

Adjuntamos la imagen con una traza de valores obtenidos por el componente electrónico.

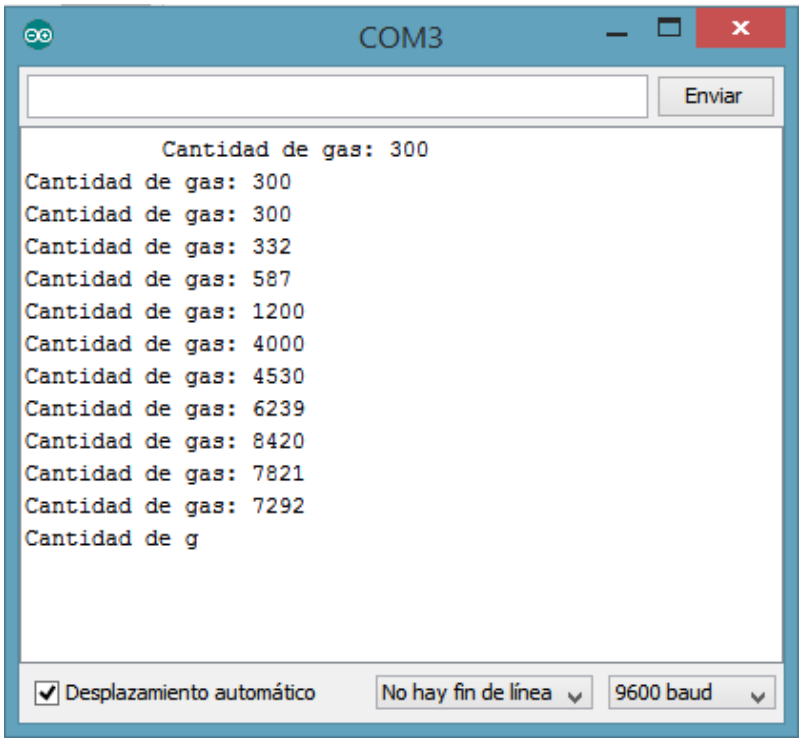


Figura 8.10: Resultados de la prueba del sensor MQ-2.

8.2.4 Sensor TMP-36

Con esta prueba deseamos comprobar el funcionamiento del sensor de temperatura TMP-36. Para ello diseñaremos el circuito entre dicho sensor y Arduino, desarrollaremos el código necesario para obtener los datos recogidos por el sensor y, por ultimo, mostraremos los resultados obtenidos.

Circuito

Mostramos, a continuación, el circuito necesario para la conexión entre Arduino y el sensor TMP-36. Debemos prestar atención a la hora de conectar el sensor a Arduino, pues es necesario conectarlo a un **pin analógico**.

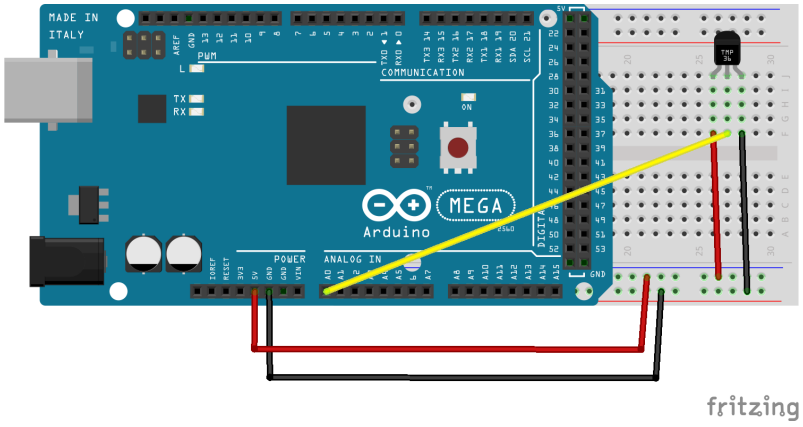


Figura 8.11: Conexión entre Arduino y sensor TMP-36.

Código desarrollado para la prueba

Presentamos el código necesario para procesar los valores que obtenemos del sensor TMP-36. Destacamos la función *analogRead()* que nos permite obtener las lecturas del sensor.

Listing 8.14: tmp36.ino

```
const int sensorPin = A0;
void setup(){
  Serial.begin(9600);
}
void loop(){
  int sensorVal = analogRead(sensorPin);
  float voltage = (sensorVal/1024.0) * 5.0;
  Serial.print("Temperatura: ");
  float temperature = (voltage - .5) * 100;
  Serial.println(temperature);
  delay(1);
}
```

Resultados de la prueba

Adjuntamos la imagen con una traza de valores obtenidos por el componente electrónico.

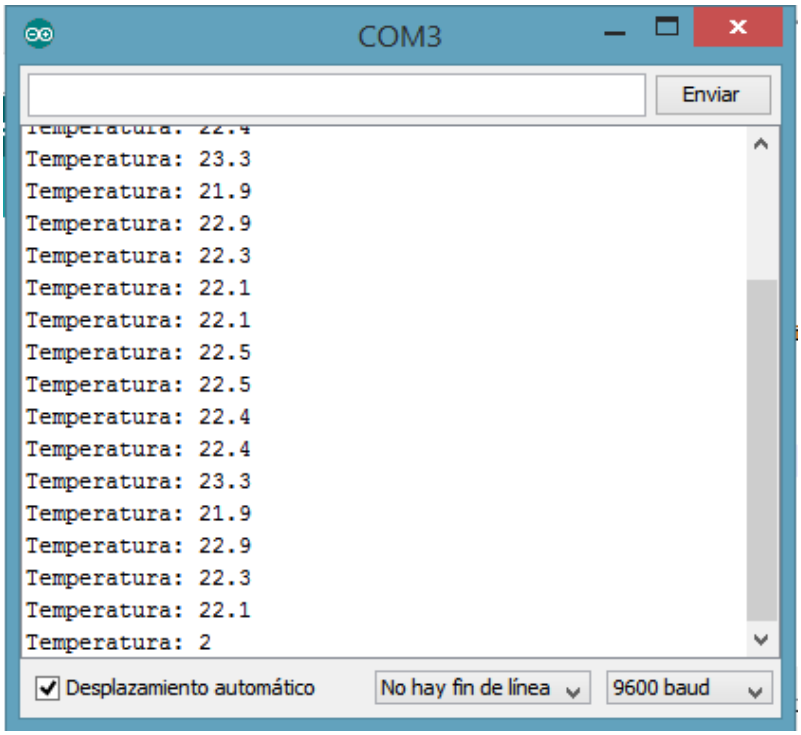


Figura 8.12: Resultados de la prueba del sensor TMP-36.

8.2.5 Puerta de garaje

En esta sección hablamos de cómo hemos simulado una puerta de garaje para nuestra maqueta. Describiremos desde su lógica de control hasta su implementación y acople físico al conjunto de los componentes que forman la maqueta.

Los componentes específicos para esta prueba que hemos usado son :

- **Motor paso a paso bipolar** (Bipolar Stepper Motor): Su definición aparece en la sección *Herramientas usadas y justificación* . Hemos decidido usar este tipo de motor por la precisión que supone en sus movimientos. Este componente lo hemos obtenido de un lector de CDs.

Figura 8.13: Motor paso a paso bipolar.

- **Módulo controlador de motores**: Para nuestro proyecto, hemos usado el *L298N Dual H Bridge Motor Drive Controller Board Module* que puede ser adquirido via Internet fácilmente y a buen precio. No es necesario este módulo para controlar el motor, pero simplifica mucho la tarea y el tamaño del circuito eléctrico resultante es mínimo. Este modulo funciona con el chip L298N y puede controlar un motor paso a paso o dos motores de corriente continua sencillos.

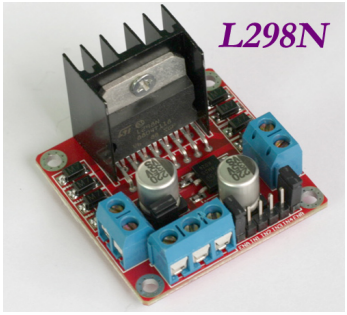


Figura 8.14: Módulo controlador L298N

Lógica de control

El comportamiento que deseamos obtener del motor es simplemente **abrir y cerrar** la puerta del garaje de la maqueta. No es una tarea sencilla tener en cuenta el esquema de pasos de un *motor paso a paso* y creemos que no es necesario tampoco incluirlo en esta memoria. Sin embargo, el lenguaje Arduino trae consigo una librería que controla este tipo de motores con facilidad y hace que su manejo sea bastante simple. La librería que hemos usado en cuestión tiene el nombre de *Stepper Library* y sirve tanto para motores unipolares como para bipolares.

Esta librería cuenta con **cuatro funciones**: *Stepper()*, que crea una instancia de un objeto Stepper que simulará el motor, *setSpeed()* que ajusta la velocidad de giro del motor en revoluciones por minuto, y por último *step()* que admite un número de vueltas que efectuará el motor. Para nuestro ejemplo hemos usado la sobrecarga del método *Stepper()* que admite 5 paramétros, ya que al ser nuestro motor de tipo bipolar, nos permitirá girar el motor en ambos sentidos usando las 4 conexiones de las que dispone. El parámetro

restante ajusta el número de pasos que componen una vuelta del motor.

Una vez tenemos creada la instancia del objeto *Stepper* podemos operar con él. Empíricamente hemos obtenido el **número de pasos** que componen una vuelta, ya que, recordemos, el motor se extrajo de un lector de DVD y no conocemos sus especificaciones. La velocidad para este motor la ajustamos en 200 rpm ya que nos ha parecido una velocidad adecuada para la apertura de la puerta del garaje. Y por último, con la función *step()* le indicamos cuantos pasos queremos hacer que gire el motor, o podemos hablar de giros completos del eje si introducimos (`Nº pasos/ vuelta X Nº vueltas`) que es la forma en que hemos decidido operar.

Al igual que para el resto de componentes de la maqueta, el formato del **comando** que recibirá la placa arduino es del tipo *Tipo-Identificador-Valor*. Por ejemplo, si queremos abrir la puerta del garaje debemos enviar el comando `M0-M01-UP` al microcontrolador, sabiendo que `M0` identifica a los componentes motorizados, `M01` identifica al motor en cuestión, y `UP` al tipo de acción.

Integración en la maqueta

Con la lógica de control ya implementada, el siguiente paso es crear el circuito e integrarlo en la maqueta. Hemos atornillado el motor al marco superior de la puerta y hemos usamos un hilo de tanza para conectar los componentes como puede verse en la figura 8.15. Para terminar, debemos tener en cuenta las leyes de la mecánica ya que la fuerza necesaria para mover la puerta es diferente a lo largo de su recorrido. Para solucionar este problema hemos añadido una polea fija que nos minimiza en gran parte este contratiempo.

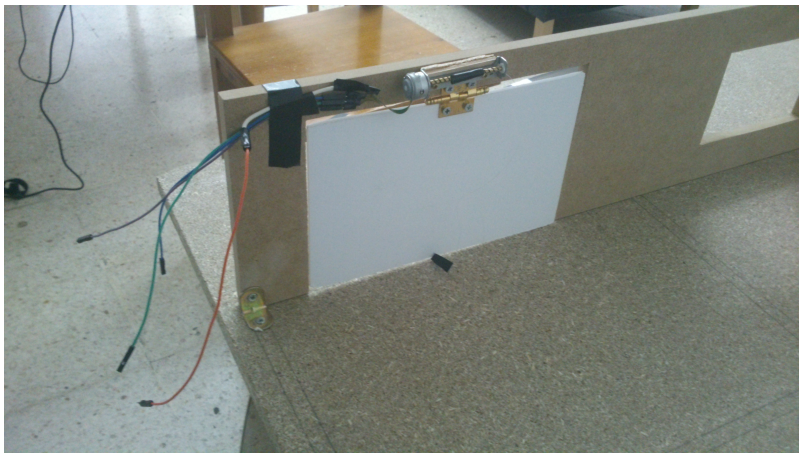


Figura 8.15: Instantánea de la puerta del garaje.

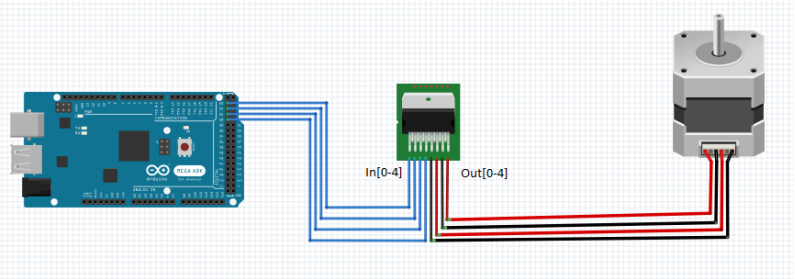


Figura 8.16: Circuito de la puerta del garaje

8.3 Pruebas de integración

8.3.1 Pruebas de estrés

Incluimos este tipo de pruebas ya que hemos de recordar que la potencia que suministra Arduino es limitada y el número de conexiones de la base de datos tampoco es demasiado alto. Tenemos bastantes componentes electrónicos conectados y se torna necesario comprobar el correcto funcionamiento más allá de las condiciones normales del sistema. Es por eso que hemos realizado una serie de pruebas en un escenario de exigencia alta.

8.3.2 Pruebas de concurrencia

Como hemos descrito en capítulos anteriores, podemos manejar nuestro sistema domótico desde las aplicaciones web y Android. Ambas realizan peticiones al servidor, ya sean de un tipo u otro, así que podemos pensar qué pasaría si un una de las aplicaciones mandase una orden al servidor alojado en la Raspberry, y la otra mandase justo la orden contraria.

Para resolver este enigma hemos probado a encender y apagar un led a la vez desde las distintas aplicaciones de control, escenario que podría darse con facilidad en un hipotético caso real. Recordemos que los comandos se almacenan en un buffer para ser transmitidos por un puerto usb hacia Arduino y además incluimos el carácter '*' para marca el fin de un comando. Con esto en mente, parece improbable una concatenación de comandos y por lo tanto esperamos un resultado positivo en este aspecto.

Los resultados de la prueba fueron satisfactorios. Hemos conseguido controlar un led concurrentemente sin aparición de fallos. Además hemos probado a modificar el estado de dos ledes diferentes en el mismo instante de tiempo con el mismo resultado. El sistema se ha comportado correctamente bajo estas condiciones y, teniendo en cuenta el alcance del proyecto, no hemos querido ir más allá y creemos que estas pruebas han sido suficientes en cuanto a concurrencia se refiere.

9 Comparación con otras alternativas

Existen numerosas opciones resultado de combinar diferentes tecnologías y protocolos con el fin de formar un sistema domótico. A continuación, presentamos aquellas que consideramos más interesantes:

- **Domótica Arduino ExControl de ExControl**: Este sistema presenta una solución muy parecida a la nuestra, basándose en componentes de bajo coste y una aplicación Android disponible para el usuario. El núcleo del sistema está basado en estos componentes:
  - **Arduino Mega 2560**: Precargado con un programa para una gestión de pines específica y el funcionamiento del módulo Ethernet.
  - **Módulo Ethernet para Arduino**: Actúa como servidor web aceptando conexiones entrantes y recibiendo los comandos desde la aplicación.
  - **Módulo reloj RTC DS2331**: Necesario para programar tareas.

  El sistema ExControl difiere del nuestro principalmente en el **servidor web**. Mientras que ExControl utiliza el módulo Ethernet, nosotros usamos Raspberry Pi con todos los beneficios que ellos conlleva, como la posibilidad de brindar una página web dinámica, registro de datos, servidor *streaming*, etc. Sin embargo, debemos lidiar con la comunicación entre Arduino y Raspberry Pi. También esta alternativa ofrece más opciones, como control por voz o programación de tareas.

- **WigWag de WigWag Inc.**: Se trata de una plataforma completa de rápida instalación que trabaja haciendo uso de múltiples sensores. La filosofía de este sistema consiste en que el usuario debe establecer una serie de **reglas** con la estructura "Si ocurre X, entonces haz Y". Estas reglas son definidas por el usuario a través de una aplicación disponible en Android y iOS. Los dispositivos hardware que forman este sistema se comunican mediante WI-FI, Z-wave o Zigbee, dependiendo del modelo. Estos son los componentes principales de la plataforma:
  - **Relé**: Dispositivo que realiza la función de maestro y puente con Internet. Se conecta automáticamente a la nube de WigWag para enviar u obtener las reglas que el usuario define. Recibe las notificaciones del dispositivo Sensor y envía órdenes a otros dispositivos para que realicen una determinada acción.
  - **Sensor**: Un conjunto de sensores integrados en una placa que en todo momento están recogiendo datos del entorno. Además, posee entradas para añadir más sensores externos y, por tanto, para crear nuevas reglas.Por último, necesita cuatro pilas AA para poder funcionar, puesto que la idea es que coloquemos este aparato donde deseemos.
  - **Tag**: Dispositivo que comunica la distancia a la que se encuentra del Relé y del Sensor. También dispone de un botón que ejecuta una regla que el usuario a configurado previamente.

La diferencia principal frente a nuestro sistema es que el **impacto de la instalación** de WigWag en un hogar u oficina es mínimo, pero su uso se reduce a un **pequeño número de sencillas tareas**, y es por ello que la compañía te ofrece un paquete en el incluye múltiples unidades de Sensor y Tag aumentando las posibilidades pero disparando el **precio del producto**, alcanzando los 650 $.

Otra gran diferencia es que con WigWag las reglas son definidas por el usuario, dando libertad a éste para crear el comportamiento del sistema. Nuestra propuesta no ofrece un sistema que podamos personalizar cada vez que queramos, sino, que otorga acciones preestablecidas. Es decir, si con WigWag necesitamos establecer una regla en la que especifiquemos que si salimos de casa, entonces apague la iluminación, con nuestro producto podemos apagar la iluminación del hogar desde cualquier lugar mediante las aplicaciones que ofrecemos.

10 Conclusiones y mejoras futuras

10.1 Conclusiones

Llegados a este punto, podemos volver la vista atrás 5 meses y enfrentarnos de nuevo al abismo que se situaba ante nosotros. Poseíamos las ganas y la capacidad pero nos amedrentaba la incertidumbre. Poco a poco íbamos resolviendo los problemas que se nos presentaban, y adquiriendo experiencia para que a día de hoy podamos enfrentarnos casi a cualquier desafío siendo más optimistas. El trabajo en grupo **no ha sido fácil**, los horarios de los miembros del equipo han tenido pocas horas en común haciendo que la coordinación entre ellos fuese una tarea imprescindible y complicando la toma de decisiones. Pero, salvando las pequeñas diferencias, el ambiente de trabajo ha sido muy favorable y nos ha motivado para seguir adelante cuando las cosas se torcían. Estamos muy orgullosos tanto del resultado final como del crecimiento personal.

En cuanto al proyecto, creemos que **hemos cumplido** con los objetivos y que hemos logrado desarrollar un producto con mucho potencial, capaz de convertirse en algo importante con una buena inversión. También, hemos llegado a la conclusión de que la gestión del proyecto ha sido vital para finalización del mismo a tiempo. Tanto el versionado del código como la revisión conjunta han facilitado mucho las tareas, y la planificación nos ha ayudado a no apartarnos del buen camino.

10.2 Mejoras futuras

El sistema desarrollado cuenta con una gran variedad de aspectos que podrían mejorarse disponiendo del tiempo adecuado. Pasamos a enumerar los que creemos cobran mayor importancia:

- En primer lugar, la idea del proyecto consistía en conseguir un sistema domótico que aprovechase la movilidad de las tecnologías a día de hoy existentes. Sin embargo, hemos creído conveniente restringir el alcance de la comunicación de las interfaces de control a un ámbito de red local, ya que exponer el sistema al público hubiese requerido de un trabajo que se aleja de los objetivos de este proyecto.
- En segundo lugar, debemos mencionar que hemos enfocado las interfaces de control a completar la funcionalidad del sistema y añadir alguna extra. Sin embargo, nos hubiese gustado aprovechar al máximo las plataformas Web y Android para darle un valor mucho mayor al proyecto. Las posibilidades son muy amplias, como por ejemplo, que la aplicación Android se adaptase a la distribución del hogar domótico con solo tener esta en una base de datos.
- Gracias a la modularidad del hardware Arduino podemos añadir más sensores y componentes electrónicos con la única preocupación de que tengamos suficien-

te potencia. Por ejemplo, podríamos añadir un modulo que incorpora **tecnología GSM**, así si no disponemos de conexión móvil, podríamos hacer que nos llegase un SMS a nuestro teléfono con alguna alerta o, también, manejar el hogar a través de mensajes de texto. Cámaras de vigilancia, reconocimiento de voz, son algunas de las posibles mejoras que nos gustaría incluir si el proyecto sigue adelante.

11 Bibliografía

Siempre que podamos seguiremos el siguiente esquema para citar las páginas web consultadas:

- Autor. Título. Lugar: fecha de edición. [fecha de consulta]. Disponible en: URL

Por otro lado, para los libros seguiremos el siguiente esquema siempre que sea posible:

- Autor. Título. Número de edición. Lugar: Editorial, año de edición. ISBN.

Libros

1. Scott Fitzgerald y Michael Shiloh. *Arduino Projects Book*. Turín: septiembre de 2012.

Páginas web

1. Sitio web oficial de Android: http://developer.android.com
2. Sitio web oficial de Arduino: www.arduino.cc
3. Sitio web elFinder: http://elfinder.org/
4. Sitio web LaTeX Stack Exchange: http://tex.stackexchange.com/
5. Sitio web Stack Overflow: http://stackoverflow.com/
6. Sitio web Wikipedia: http://es.wikipedia.org
7. eLinux. *RPi USB Wi-Fi Adapters*. [Consulta: 6 de febrero de 2014]. Disponible en: http://www.elinux.org/RPi_USB_Wi-Fi_Adapters
8. Timothy L. Warner. *Raspberry Pi: I'm Ready to Get Started. Now What?*. QUE: 4 de diciembre de 2013.[Consulta: 7 de febrero de 2014]. Disponible en: http://www.quepublishing.com/articles/article.aspx?p=2164581
9. John Browning. *So you got a Raspberry Pi: now what?*. Engadget: 4 de septiembre de 2012. [Consulta: 10 de febrero de 2014]. Disponible en: http://www.engadget.com/2012/09/04/raspberry-pi-getting-started-guide-how-to/
10. *Raspberry PI headless installation*. Blog Confessions of a forgetful programmer: 3 de junio de 2013. [Consulta: 10 de febrero de 2014]. Disponible en: http://forgetfulprogrammer.wordpress.com/2013/06/03/raspberry-pi-headless-installation/
11. Mario Pérez. *Actualizar nuestra Raspberry Pi con Raspbian*. Geeky Theory: 15 de diciembre de 2013. [Consulta: 11 de febrero de 2014]. Disponible en: http://geekytheory.com/actualizar-nuestra-raspberry-pi-con-raspbian/

12. Mario Pérez. *Arduino /+ Raspberry Pi - Raspduino* . Geeky Theory: 30 de julio de 2013. [Consulta: 20 de febrero de 2014]. Disponible en: http://geekytheory.com/arduino-raspberry-pi-raspduino/

13. MRCELHW. *Tutorial Raspberry Pi - Crear servidor web* . Geeky Theory: 10 de noviembre de 2013. [Consulta: 21 de febrero de 2014]. Disponible en: http://geekytheory.com/tutorial-raspberry-pi-crear-servidor-web/

14. *PHP/Apache* . IBEX. [Consulta: 23 de febrero de 2014]. Disponible en: http://www.raspberry-projects.com/pi/software_utilities/phpapache

15. darkaw_remse. *Raspberry+Arduino=Raspduino* . Blog Moleros Geek: 8 de febrero de 2014. [Consulta: 24 de febrero de 2014]. Disponible en: http://morelosgeek.com/2014/02/raspberryarduinoraspduino/

16. *The history of Smart Homes* . Smart Homes [Consulta: 28 de febrero de 2014]. Disponible en: http://www.smart-homes.nl/Smart-Homes/Geschiedenis.aspx?lang=en-US

17. Alec Ryan. *Tutorial: store Arduino data with RaspBerry PI to MySql*. Daviom: 22 de abril de 2013. [Consulta: 25 de abril de 2014]. Disponible en: http://www.daviom.com/tutorial/tutorial-store-arduino-data-with-raspberry-pi-to-mysql/

18. Regata. *Sensor dht11 (humedad y temperatura) con arduino*. Daviom: 24 de abril de 2013. [Consulta: 28 de marzo de 2014]. Disponible en: http://tallerarduino.com/2012/12/24/sensor-dht11-humedad-y-temperatura-con-arduino/

19. Creately. *Flowchart Symbols and their usage*. Creately. [Consulta: 27 de abril de 2014]. Disponible en: http://creately.com/diagram-type/objects/flowchart

20. Steve Breuning. *Run a script on start up*. Blog Raspberry Web Server: 22 de octubre de 2013. [Consulta: 10 de mayo de 2014]. Disponible en: http://raspberrywebserver.com/serveradmin/run-a-script-on-start-up.html

A Configuración de Raspberry Pi

Los componentes necesarios para la puesta en marcha y configuración del dispositivo Raspberry Pi son los siguientes:

- Minicomputador *Raspberry Pi*
- Tarjeta de memoria SD (Recomendamos una capacidad mayor de 4 GB y Class 6 o superior)
- Cargador (Recomendamos de una intensidad de 1200 mA y un voltaje de salida de 5 V)

A.1 Formato SD e instalación de sistema operativo

Para empezar, debemos descargar el sistema operativo que deseamos instalar. En nuestro caso hemos optado por usar Raspbian debido a que está optimizado para Raspberry Pi y la mayor parte del software del dispositivo se encuentra disponible para este sistema. Una vez descargado el sistema procedemos a formatear la tarjeta SD desde un ordenador y guardar la imagen del sistema descargado en ella. A continuación, mostramos cómo instalar Raspbian en la SD mediante el programa Win32DiskImager para Windows.

1. Introducimos la tarjeta SD en el lector.
2. Ejecutar Win32DiskImager. Nos aparece la siguiente pantalla:

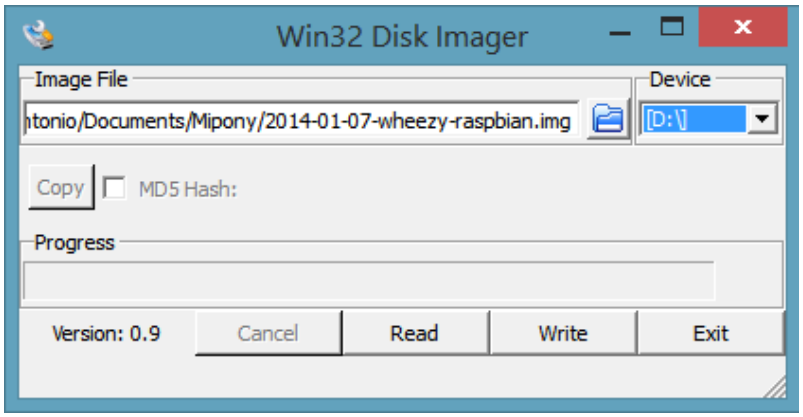


Figura A.1: Intefaz de Win32DiskImager.

3. En el campo Image File, seleccionamos la imagen del sistema operativo que previamente se ha descargado.
4. Seleccionamos la letra correspondiente a la unidad del lector de tarjetas SD en el campo Device.
5. Por último, pulsamos sobre Write y esperamos a que termine el proceso.

Una vez completado el anterior paso, ya podemos insertar la tarjeta SD en la Raspberry Pi y conectarla al cargador para que inicie automáticamente Raspbian.

A.2 Interfaz gráfica de Raspbian

Para poder interactuar con Raspberry Pi necesitamos conectar el dispositivo a una pantalla, además de teclado y ratón. Pero en nuestro caso, hemos decidido evitar comprar más materiales y poder comunicarnos con Raspberry Pi simplemente mediante una conexión a un servidor VNC alojado en el minicomputador. Para ello seguiremos los siguientes pasos:

1. Establecemos una conexión SSH (en área local) entre la Raspberry Pi y un ordenador u otro dispositivo. Por defecto, Raspberry Pi tiene habilitado el servidor SSH así que solo necesitamos un cliente SSH para poder establecer la comunicación. En esta ocasión hemos usado el cliente PuTTy.

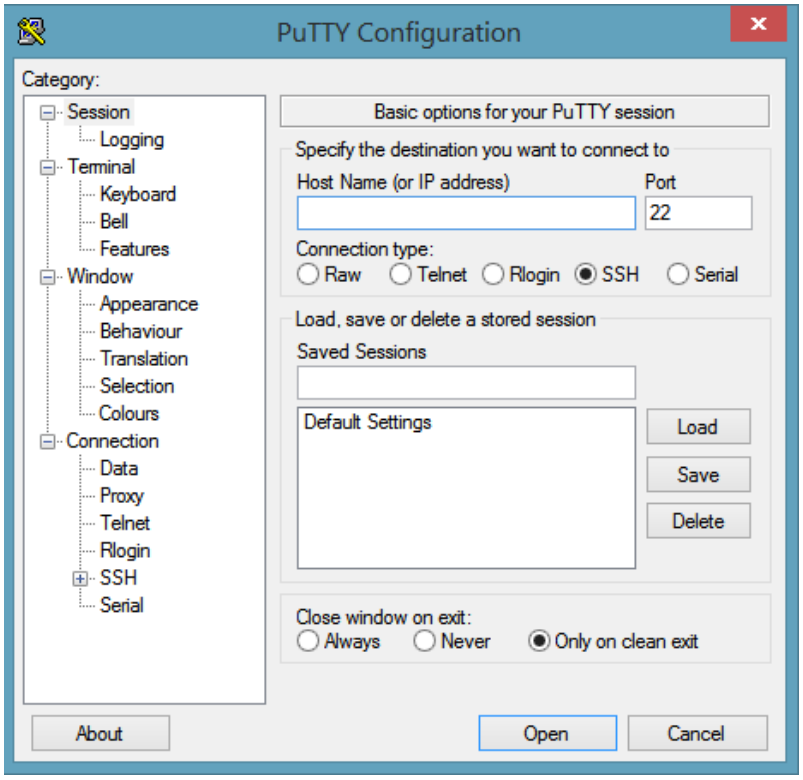


Figura A.2: Interfaz de PuTTy.

2. Especificamos la IP de Raspberry Pi en el campo Host Name (or IP address) y pulsamos sobre Open.

3. Establecida la conexión SSH, tenemos que introducir el usuario y clave de la Raspberry Pi.

   - Usuario: pi
   - Contraseña: raspberry

4. Procedemos a instalar el servidor VNC (TightVNC Server) en la Raspberry introduciendo el siguiente comando:

   - *sudo apt-get install tightvncserver*

5. Establecemos una contraseña a nuestro servidor VNC. En este caso usamos "arduino" como contraseña:

   - *vncpasswd arduino*

6. Inciamos el servicio VNC mediante el comando:

   - *tightvncserver*

7. Ya está todo listo para acceder al entorno gráfico de la Raspberry Pi mediante el cliente TightVNC Java Viewer introduciendo su IP y puerto.

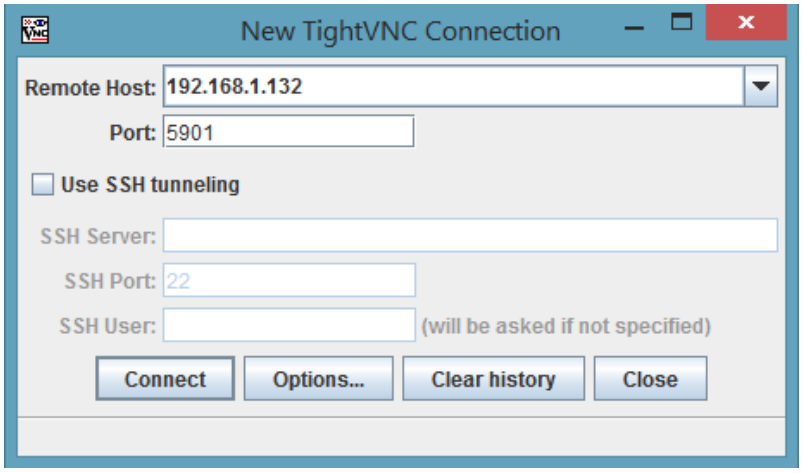


Figura A.3: Interfaz de TightVNC Java Viewer.

Hemos querido, para ahorrar tiempo, que se inicie el servidor VNC cada vez que se encienda la Raspberry Pi. Para que el servidor se ejecute automáticamente en el arranque añadimos al archivo `/etc/rc.local` la siguiente línea de comando:

- *su -c /usr/bin/tightvncserver"pi*

Si deseamos cambiar la resolución de la pantalla así como la profundidad de color, modificamos la anterior línea añadiendo lo siguiente:

- *su -c /usr/bin/tightvncserver -geometry 1024x720 -depth 16"pi*

A.3 Actualización y puesta a punto

Es probable que el software de la Raspberry Pi no se encuentre actualizado. Para resolver este problema ejecutamos los siguientes pasos, quedando así el dispositivo listo para su uso:

- Actualizamos los repositorios: *sudo apt-get -y update*

- Actualizamos todos los programas instalados: *sudo apt-get -y upgrade*
- Actualizamos el kernel: *sudo rpi-update*

A.4 Servidor FTP

Raspberry Pi no es un dispositivo potente, por tanto, la tarea de programar mediante su interfaz gráfica puede ser muy lenta y llegar a desesperarnos. Para solventar este problema podemos programar en nuestros ordenadores personales y pasar los archivos a la carpeta que deseemos mediante el protocolo FTP. En nuestro caso hemos elegido el servidor vsftpd desarrollado por Chris Evans y con licencia GPL. Pasamos entonces a detallar los pasos necesarios para instalar el servidor FTP en Raspberry Pi:

1. Instalamos el servidor mediante el siguiente comando:
   - *sudo apt-get install vsftpd*
2. Abrimos el archivo de configuración del servidor FTP con el comando:
   - *sudo nano /etc/vsftpd.conf*
3. Navegamos por el archivo hasta encontrar y cambiar las siguientes líneas:
   - *anonymous_enable=YES* la cambiamos a *anonymous_enable=NO*
   - *#local_enable=YES* la cambiamos a *local_enable=YES*
   - *#write_enable=YES* la cambiamos a *write_enable=YES*
4. Por último añadimos al final del archivo la siguiente línea y después lo guardamos:
   - *force_dot_files=YES*
5. Reiniciamos el servidor FTP:
   - *sudo service vsftpd restart*
6. Damos permisos a la carpeta donde deseamos enviar los archivos que programamos.
   - *sudo chmod 0777 -R 'directorio_carpeta'*

Con esto ya tenemos configurado el servidor FTP. Ahora podemos conectarnos a él con un cliente FTP introduciendo:

- La dirección IP de Raspberry Pi
- El usuario de Raspbian
- La contraseña de usuario de Raspbian
- El puerto: 21

A.5 Servidor web Apache

En este proyecto hemos instalado el servidor web Apache 2 en Raspberry Pi. Además del servidor, también hemos instalado el módulo necesario para la integración entre nuestro servidor web y el lenguaje PHP, todo ello con el siguiente comando:

- *sudo apt-get install apache2 php5 libapache2-mod-php5*

Despues reiniciamos el servidor:

- *sudo service apache2 restart*

Por último, el directorio para almacenar los archivos necesarios para el diseño web es `/var/www`.

A.6 Base de datos MySQL

En nuestro proyecto decidimos instalar una base de datos MySQL en Raspberry Pi para almacenar datos, como aquellos recogidos por los sensores, notificaciones de Arduino, credenciales del usuario, etc. Para poder instalar el servidor MySQL en Raspberry Pi y hacer uso de esta herramienta debemos seguir los siguientes pasos:

1. Instalamos el servidor MySQL y el cliente MySQL con el siguiente comando:
   - sudo apt-get install mysql-server mysql-client
2. Durante la instalación, debemos introducir la contraseña que deseamos asignar al usuario *root*, usuario por defecto del servidor MySQL con el que accederemos al mismo. Dicha contraseña no debe de ser la misma que la contraseña del usuario administrador del sistema operativo de Raspberry Pi.
3. Cuando finalice la instalación estará todo listo y podremos acceder al servidor MySQL mediante el siguiente comando:
   - *mysql -u root -p*
4. Por último, para crear una base de datos cuyo nombre sea, por ejemplo *raspduino*, debemos teclear lo siguiente:
   - *create database raspduino*

A.7 Cuotas de disco

Cuando trabajamos ofreciendo un servicio de almacenamiento de información, sea del tipo que sea, debemos administrar nuestro sistema acorde a los recursos que nos brinda. Independientemente del número de usuarios que tengamos, es muy recomendable fijar una porción de nuestros recursos, en este caso de disco duro, para cada usuario a favor de la **escalabilidad** y **disponibilidad** del sistema.

Una cuota de disco es una restricción de espacio de disco duro con el fin de gestionar de una manera eficiente los recursos del sistema. Existen dos tipos de cuota de disco:

1. **Cuota de bloques**: Espacio dedicado medido en número de bloques del sistema de archivos. El sistema guarda los archivos en bloques; cada bloque tiene una tamaño distinto dependiendo del tipo de sistema, por ejemplo en Raspbian un bloque tiene una tamaño de 512 KB.

2. **Cuota de inodo**: Espacio dedicado medido en número de inodos del sistema de archivos. Un inodo es una estructura de datos que apunta a los bloques que almacenan los datos del archivo al que hace referencia. Además, esta estructura también almacena información sobre permisos, fechas, propietarios, etc. Los inodos se usan en sistemas operativos tipo UNIX.

En nuestro proyecto, hemos usado las cuotas de disco para que el usuario disponga de 1 GB de la tarjeta SD de la Raspberry Pi con el fin de que almacene en dicho espacio los archivos de música que desee para después reproducirlos mediante la aplicación web. Para ello hemos seguido una serie de pasos que, a continuación, explicamos.

A.7.1 Implementación de cuotas de disco

Cada sistema operativo tiene una forma de gestionar las cuotas de disco. Como usamos Raspbian, la implementación está orientada a sistemas Linux. En resumen, implementaremos cuotas de disco usando un sistema de archivos virtual, dicho sistema lo crearemos mediante un archivo de disco.

Software requerido

Antes de comenzar, debemos instalar en Raspberry Pi la herramienta que nos ayudará a crear y gestionar las cuotas de disco. Dicha herramienta se llama ***quotatool*** y puede ser instalada en Raspbian mediante el siguiente comando:

- *sudo apt-get install quotatool*

Una vez instalado podemos proceder a crear el sistema de archivos virtual descrito a continuación.

Creación de un sistema virtual de archivos

Con los siguientes pasos vamos a crear un sistema virtual de archivos *ext3* montado en el directorio que deseamos limitar su tamaño. En nuestro caso, dicho directorio es `/var/www/player/media/songs` y lo limitaremos a 1 GB.

1. Como administrador, creamos un punto de montaje de cuota, es decir, el directorio que deseamos limitar:

   - *mkdir -p /var/www/player/media/songs*

2. Después, creamos un archivo con el tamaño de la limitación (1 GB) en el lugar que prefiramos, por ejemplo en `/usr/disk-img`:

   - *mkdir -p /usr/disk-img*
   - *dd if=/dev/zero of=/usr/disk-img/disk-quota.ext3 count=2097152*

Mediante el comando de arriba creamos un archivo de 1 GB debido a que un bloque de disco duro es 512 B. Luego, $2097152 * 512 = 1073741824$ B. Podemos confirmar este tamaño mediante el siguiente comando:

- *ls -lh /usr/disk-img/disk-quota.ext3*

3. Ahora, procedemos a formatear el sistema de archivos:

   - */sbin/mkfs -t ext3 -q /usr/disk-img/disk-quota.ext3 -F*

   El campo *-t* se refiere al tipo de sistema de archivos, *ext3*. El campo *-q* se refiere al dispositivo y *-F* fuerza a realizar el proceso sin mostrar ninguna advertencia durante la ejecución.

4. Para habilitar el sistema de archivos cada vez que se reinicie Raspberry Pi, añadimos la siguiente línea en `/etc/fstab`

   - */usr/disk-img/disk-quota.ext3 /var/www/player/media/songs ext3 rw,loop,usrquota,grpquota 0 0*

5. Montamos el sistema de archivos:

   - *mount /var/www/player/media/songs*

   Cuando termine el proceso, podemos observar que aparece la carpeta `lost+found` en el directorio `/var/www/player/media/songs`. Desde ahora, podemos añadir los archivos que deseamos.

Cuotas y usuarios

Asignamos la cuota al usuario que deseemos. En nuestro caso, es el usuario www-data que pertenece al servidor web Apache.

1. Como administrador, verificamos que el sistema de archivos soporte las cuotas:

   - *quotacheck -cug /var/www/player/media/songs*

   Automáticamente, el sistema ha creado dos archivos: *aquota.group* y *aquota.user*.

2. Mediante el siguiente comando añadimos www-data a la cuota deseada:

   - *edquota -f /var/www/player/media/songs www-data*

3. En el archivo de texto que aparece establecemos el número de bloques o inodos que como máximo podrá usar el usuario www-data.

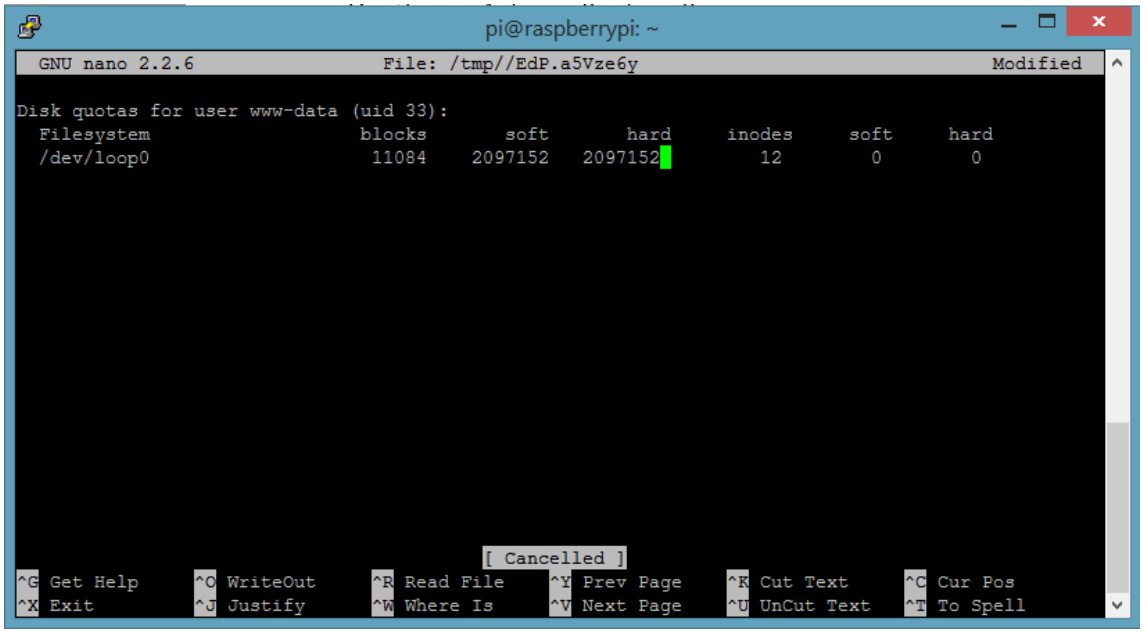


Figura A.4: Número de bloques para el usuario www-data.

4. Por último, activamos las cuotas con el siguiente comando:

   - *quotaon /var/www/player/media/songs www-data*

B Entorno de desarrollo de Arduino

Explicamos a continuación como instalar y configurar el entorno para Arduino. Recomendamos descargar la última versión estable del software oficial, para este caso hemos elegido la versión 1.0.5 correspondiente al sistema operativo Windows. Una vez descargado, debemos seguir los pasos del asistente de instalación. En la mayoría de los casos no hace falta cambiar nada de la configuración por defecto. Y ya tenemos listo el entorno de desarrollo Arduino.

La interfaz que presenta el entorno está formada por una serie de menús, una barra de herramientas, un editor de texto (donde se escribirá el código del programa que se desea cargar), una área de notificaciones y una consola de texto. Ver firgura B.1

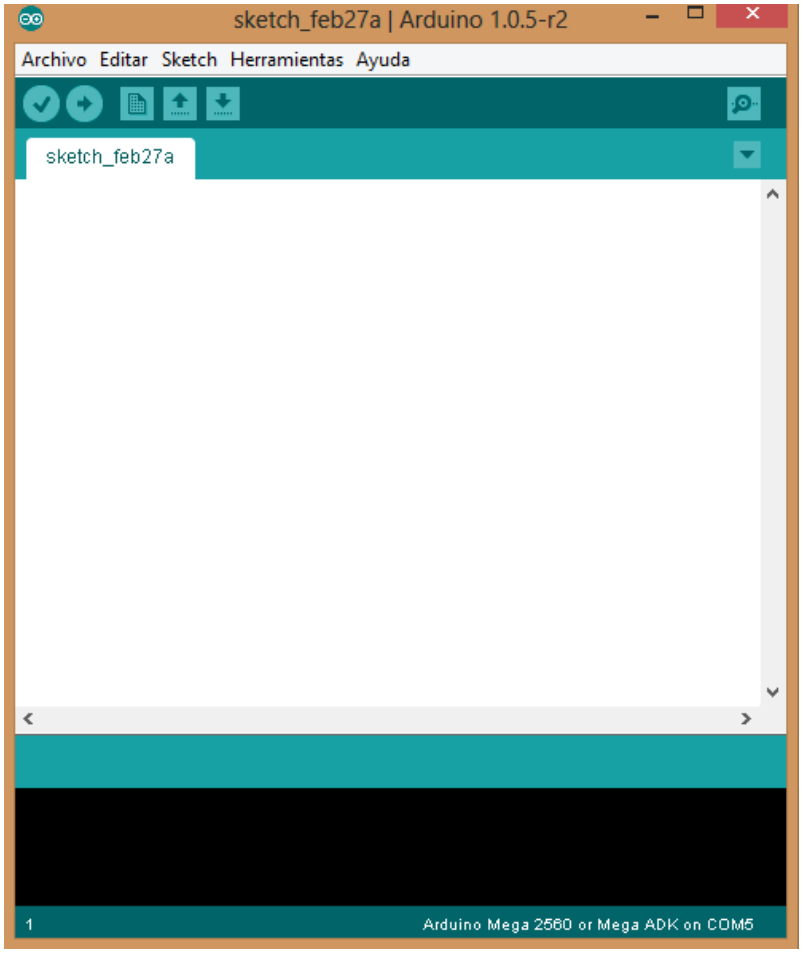


Figura B.1: Presentación del entorno de desarrollo.

B.1 Menús

De esta parte del entorno cabe destacar el menú desplegable de Herramientas donde debemos seleccionar qué placa Arduino estamos utilizando y el puerto serie en el que

está conectada dicha placa. En este caso, seleccionamos la opción Arduino Mega 2560 or Mega ADK para la placa y COM4 para el puerto. Figura B.1

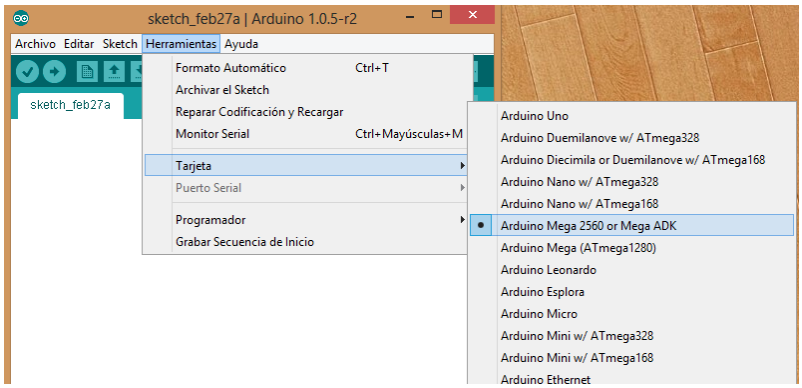


Figura B.2: Seleccionamos la versión adecuada de la placa.

B.2 Barra de herramientas

Está compuesta por botones de acceso rápido disponibles también en los menús. Los botones son los siguientes:

- Verificar: comprueba y compila el código.
- Cargar: cumple la misma función que Verificar y además sube el código a la placa de Arduino.
- Nuevo: crea un nuevo sketch.
- Abrir: para abrir un sketch anteriormente guardado.
- Guardar: guarda el sketch actual.
- Monitor Serial: abre una ventana que sirve de comunicación con Arduino. Desde ella podemos obtener datos de la placa, así como enviar datos a la misma.

B.3 Editor de texto

Área destinada a escribir el programa que deseamos que ejecute Arduino. Dicho programa siempre debe constar de 2 partes.

La primera parte es el método *setup()* donde inicializamos variables y definimos el estado de los pines, entre otras cosas. Esta función sólo se ejecuta una vez.

La segunda parte está formada por el método *loop()* que se ejecuta sucesivamente y es donde añadimos el código correspondiente a las acciones que deseamos que Arduino realice. Al ejecutarse en bucle permite al programa adaptarse a la situación y poder responder adecuadamente.

B.4 Área de notificación

Muestra la situación tras utilizar los botones de acceso directo de Verificar y Cargar

B.5 Consola de texto

Especifica con mayor detalle los mensajes del Área de notificación.

C Contenido del DVD adjunto

En el DVD adjuntamos las siguientes carpetas y archivos:

- **Código fuente**: Código desarrollado para las plataformas Android, Arduino y Web. Además, incluimos el código fuente de la memoria en LaTeX, para compartirlo con la comunidad universitaria.
- **Circuitos**: Circuitos diseñados para los componentes electrónicos y eléctricos del proyecto. Los incluimos en formato *.fzz* (Fritzing).
- **Maqueta**: Adjuntamos en esta carpeta, contenido multimedia acerca del proyecto. Imágenes y vídeos.
- **Memoria en formato PDF**: Memoria en formato PDF.
- **Tareas**: Tareas realizadas en el proyecto desglosadas por componente del mismo.