UNIVERSIDAD
REY JUAN CARLOS

GRADO EN INGENIERÍA DE SISTEMAS DE TELECOMUNICACIÓN

Curso Académico 2014/2015

Trabajo Fin de Grado

# IMPLEMENTACIÓN DE SISTEMAS DE CONTROL SOBRE RASPBERRY PI

Autor : Jaime Amigueti Fernández de Bobadilla

Tutor : Dr. Gregorio Robles Martínez

# Trabajo Fin de Grado

IMPLEMENTACIÓN DE SISTEMAS DE CONTROL SOBRE RASPBERRY PI

**Autor :** Jaime Amigueti Fernández de Bobadilla

**Tutor :** Dr. Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 2015, siendo calificada por el siguiente tribunal:

**Presidente:**

**Secretario:**

**Vocal:**

y habiendo obtenido la siguiente calificación:

**Calificación:**

Fuenlabrada, a de de 2015

# Dedicatoria

*Dedicado a mi familia.*

*Gracias por vuestra infinita paciencia.*

# Resumen

Desde su aparición en escena, placas como `Arduino` o `Raspberry Pi` han redefinido por completo la actitud de los usuarios medios por los circuitos empotrados. Dado su bajo coste, sencillez y sus múltiples posibilidades, estos elementos se han convertido en una pieza muy popular dentro de la comunidad online de programadores.

Existe multitud de material informativo al respecto en internet sobre proyectos o sobre como iniciarse en trabajos propios: cursos, foros, tutoriales, etc. A través de ellos se puede uno hacer a la idea del entusiasmo que producen este tipo de tecnologías.

Otra de las tecnologías de moda actualmente son las 'redes inalámbricas de sensores' y el 'internet de las cosas'. La primera se basa en la monitorización de parámetros y su posterior traslado a la nube. Y la segunda por dotar de inteligencia electrónica a elementos cotidianos de nuestro entorno y a su interconexión a través de internet.

Este proyecto intenta introducirse en estos tres mundos. Para ello se ha diseñado un sistema que permita obtener datos a través de los sensores conectados a una `Raspberry Pi`. También se dota al sistema de las herramientas necesarias para manipular otro tipo de circuitos eléctricos tales como relés, motores, etc. Todo esto se presenta al usuario en forma de aplicación web que además de intuitiva, facilita su uso.

# Summary

Since their appearance on scene, electronics such as `Arduino` or `Raspberry Pi` have completely refefined the attitude of average users to embedded systems. Due to its low price, simplicity and multiple possibilities, this electronics have become a very popular piece among the online community of programmers.

There is plenty of informative material about it on the internet about projects or how to get started in our own work: courses, forums, tutorials, etc. Through them you can realize the excitement these technologies produce.

Other fancy technologies now a days are 'wireless network sensors' and the 'internet of things'. The first is based on monitoring parameters and then transferring them to the cloud. The second one by providing electronic intelligence to everyday elements of our environment and interconnecting them via the internet.

This project attempts to enter these three worlds. We have designed a system that allows the acquisition of data through sensors connected to a `Raspberry Pi`. It also provides the system with tools to manipulate other electrical circuits such as relays, motors, etc. All this is presented to the user as an intuitive web application easy to use.

# Índice general

# Índice de figuras

# Índice de cuadros

# Capítulo 1

# Introducción

Se ha diseñado un sistema electrónico que, a través de sensores, recopile información y la aloje en la nube. Para hacer el sistema más completo también se dota de funcionalidad, permitiendo al usuario encender y apagar circuitos o programar el funcionamiento de los mismos de forma remota.

Para ello usaremos una `Raspberry Pi` como base del proyecto. A ella le iremos añadiendo distintos componentes electrónicos como sensores, interruptores, multiplexores, etc. Al final tendremos un dispositivo autónomo capaz de tomar medidas del entorno así como de actuar en consecuencia.

Otra de las cosas que se ha perseguido es implicarse, al igual que muchos usuarios de la comunidad online, en un proyecto del tipo DIY, 'Do It Yourself'[1]. La filosofía DIY consiste en eso mismo, en ser el protagonista de nuestra construcciones, reparaciones y 'chapuzas'.

Como podemos comprobar no es algo que se limite al mundo de la electrónica sino que se extrapola a cualquier ámbito de nuestra vida cotidiana.

Volviendo al mundo digital, los componentes electrónicos han sido material reservado para las grandes riquezas o instituciones como universitarias, investigadoras o militares; debido a su elevado coste. Afortunadamente esto ya no es así. La tendencia ha propiciado un continuo

[1] 'Hágalo Usted Mismo' en castellano.

descenso de precios haciendo más accesibles estos mismos componentes.

En un momento dado aparecen en escena (muy probablemente en la universidad MIT[2] entre las décadas de 1950-1960) un grupo de inquietos, ¿los primeros *hackers*? No se deje engañar por la mala connotación que ha adquirido esta palabra con los años. Los *hackers* hacen *hacks* que no son más que 'soluciones elegantes para problemas importantes' [25]. Esta gente son los auténticos pioneros del software y el hardware libre.

Hace ya muchos años que el precio de los componentes electrónicos se ha visto severamente reducido. Esto ha provocado que la comunidad se haya implicado. La aparición de elementos como `Arduino` o `Raspberry Pi` han democratizado el DIY de la electrónica. ¿Qué implica esto? Que ya no solo podemos encontrar en la red software libre, sino que cada vez vemos más ejemplos de hardware libre. Por tanto, paulatinamente podemos encontrar más esquemáticos de circuitos de libre uso y modificación. Pero no son solo los esquemáticos, también podemos encontrar miles de manuales para fabricarnos casi cualquier cosa. Gracias a ello gozamos de todos los productos que podemos necesitar para realizarnos como personas y encima, a un coste generalmente asumible.

## 1.1. Descripción del Problema

Desde hace una serie de años las ventas de los ordenadores se han visto muy mermadas. Esto es debido a la aparición de otro dispositivos tales como 'tabletas' o 'smartphones'. Las continuas mejoras en hardware y electrónica han permitido la aparición de dispositivos de menor tamaño que permiten al usuario disfrutar del mismo contenido pero con un mayor grado de comodidad y libertad. Lamentablemente la mayoría de estos sistemas son 'empotrados' lo que dificulta mucho su modificación.

Una vez salidos al mercado productos como `Arduino`, `Raspberry Pi` o `Beaglebone` los usuarios empiezan a descubrir nuevos sistemas de bajo coste y altas prestaciones totalmente libres (o liberados en su mayoría). Esto desata el auge dentro de la comunidad. Si ojeamos en

[2]Instituto Tecnológico de Massachusetts.

la web encontraremos millones de proyectos en foros y blogs; los hay más interesantes que otros. Sin embargo, todos sacan lo mejor de sus autores: ganas de aprender, de desarrollar y de compartir.

Intentando emular los méritos de quienes me precedieron. Se pretende desarrollar un sistema que tenga la capacidad de tomar decisiones gracias a datos qué el mismo recoge. Para ello nos apoyaremos mucho en la electrónica. Dado que la `Raspberry Pi` no cuenta con sensores será necesario desarrollar nuestros propios circuitos.

El principal objetivo de este proyecto es el de poder recrear un trabajo sencillo el cual, quizás, pueda asentar las bases de futuros proyectos de mayor calado. Obviando la naturaleza de la información recopilada, en este proyecto se crea una plataforma donde el usuario puede añadir componentes electrónicos al sistema y desde la nube tener el control de los mismos.

## 1.2. Objetivos

Una vez zanjado el objetivo del proyecto es necesario limitar unas metas que nos permitan llevarlo a cabo. Para este proyecto, dichas metas son:

1. *Diseño e implementación de circuitos.*

   Antes de nada es fundamental realizar varios circuitos. Así se irá adquiriendo la experiencia necesaria antes de manipular la `Raspberry Pi`. Además, existen multitud de ideas con las que podemos entretenernos en esta parte del proceso.

2. *Diseño del código de la electrónica.*

   Como cada circuito es único, ya sea por su carácter analógico o digital, es necesario desarrollar varios *script* que permitan a la `Raspberry Pi` manipular la electrónica. Así, es probable que tengamos código reutilizable pero por lo general será necesario desarrollar un *script* por cada circuito instalado.

3. *Diseño del servidor* `Django`.

   Para hacer el proyecto más manejable por el usuario se decidió crear varias aplicaciones web. `Django` ofrece las herramientas necesarias para desarrollar dichas funciones.

Además cuenta con un servidor web.

4. *Implementar todas las funciones del servidor.*

   Poco a poco iremos añadiendo funciones al servidor. Es decir, desarrollar un conjunto de herramientas que doten a nuestro proyecto de la mayor autonomía. Entre estas herramientas podemos identificar programadores, actuadores, etc.

5. *Diseño del entorno con* `Bootstrap`.

   Una vez otorgada toda la funcionalidad al servidor es necesario dotarle de un entorno visual agradable cara al usuario. Para ello se recurrirá al framework `Bootstrap`.

## 1.3. Estructura de la memoria

Esta memoria está dividida en cuatro capítulos incluyendo este primero introductorio.

1. *Introducción*

   En este primer capítulo hemos explicado el problema, la mativación que nos mueve a abordarlo y los objetivos marcados para su desarrollo.

2. *Estado del Arte*

   A continuación describimos la actualidad de la tecnología usada en el desarrollo de la aplicación y esta misma memoria. Se hace especial hincapié en la descripción de la `Raspberry Pi` pués es el eje central de todo el proyecto.

3. *Implementación y Desarrollo del Proyecto*

   En este capítulo se describe la aplicación diseñada. Se mencionan los módulos que la componen, su estructura y su funcionamiento. Al final del capítulo se describen un par de aplicaciones desarrollas con las herramientas implementadas.

4. *Conclusiones*

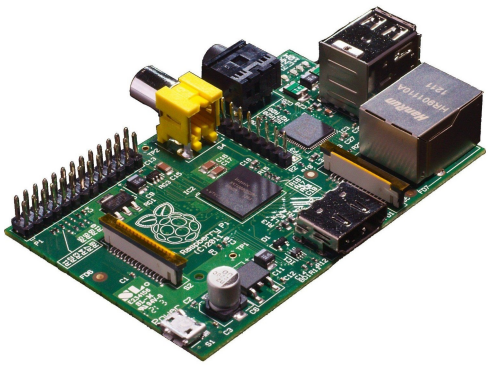
   Al final de esta memoria se encuentran las conclusiones extraidas tanto del proyecto como de la `Raspberry Pi`. También se hace un repaso de los conocimientos necesarios así como los aprendidos.

# Capítulo 2

# Estado del Arte

## 2.1. Raspberry Pi

La `Raspberry Pi` es un mini ordenador del tamaño de una tarjeta de crédito lanzado al mercado el 29 de febrero de 2012. Nace de la fundación británica con el mismo nombre para fomentar la enseñanza de 'ciencias de la computación' en escuelas públicas. Aunque no consigue el éxito esperado en este sector, sí que tiene una fantástica acogida en las escuelas privadas y entre los millones de desarrolladores y usuarios de la comunidad online, superando las 2 millones de ventas a comienzo del 2014[1].



Cuadro 2.1: Raspberry Pi [1]

A pesar de su reducido tamaño y peculiar arquitectura, la `Raspberry Pi` es un ordenador totalmente funcional que no requiere más que un monitor, un par de periféricos y una tarjeta SD donde alojar el sistema operativo y los datos.

Después de su lanzamiento, apareció un nuevo modelo llamado 'rev. B', que mejoraba y sustituía a su antecesor la 'rev A'. No existen

[1] ¡Cinco millones en febrero de 2015!

grandes diferencias entre un modelo y otro. La segunda versión duplica el número de puertos USB, la memoria RAM y añade un puerto Ethernet[2].

### 2.1.1. Raspberry Pi Hardware

La `Raspberry Pi` cuenta con una curiosa arquitectura. Incluye un chip SoC (System on a Chip) Broadcom *BCM2835*. En el corazón de este sistema encontramos un procesador *ARM 1176JZF-S* con un reloj a 700MHz, aunque algunos usuarios ya han sobrepasado con creces este límite.

Este tipo de procesadores tienen una arquitectura RISC de 32 bits. Su simplicidad, unido a su reducido consumo, los hacen ideales para el mercado de le electrónica móvil e integrada. De hecho, en 2005, entorno al 98 % de los teléfonos móviles vendidos incluían al menos un procesador ARM. Sin embargo, la arquitectura escogida está basada en la versión seis de ARM. Esto lo hace incompatible con muchos sistemas operativos.

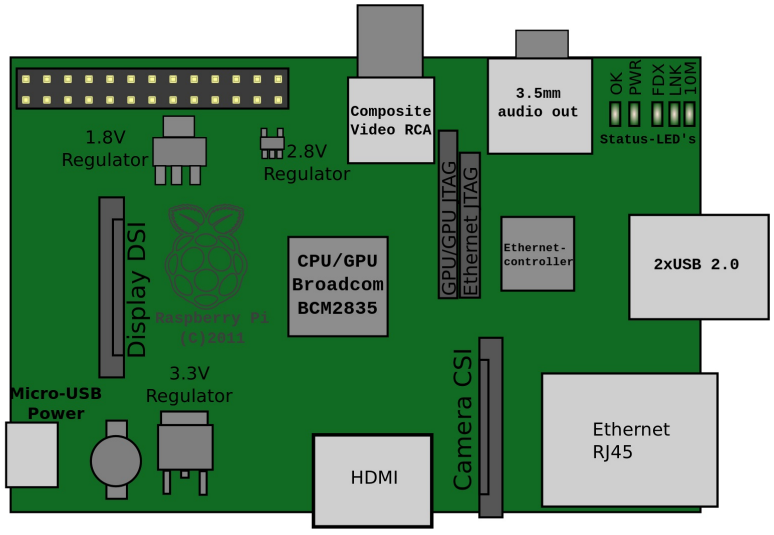



Figura 2.1: Raspberry Pi Hardware [2]

El sistema Broadcom está diseñado, según su fabricante, para ofrecer excelentes resultados en

[2]Tras la edición y antes de la publicación de esta memoria, aparecieron en el mercado varios modelos nuevos de `Raspberry`: modelo A+, B+ y `Raspberry` 2 B

aplicaciones multimedia a full HD a un precio económicamente y energéticamente reducido.

Para reducir costes, la `Raspberry Pi` no incluye un reloj en tiempo real. Por eso se ve en la necesidad de sincronizarse de forma externa. Además en su segunda versión se incorporó un conector Ethernet que se alimenta a través de USB.

| | rev A | | rev B |
|---|---|---|---|
| SOC | | Broadcom BCM2835 | |
| CPU | | ARM 1176JZF-S a 700 MHz | |
| Juego de Instrucciones | | RISC de 32 bits | |
| GPU | | Broadcom VideoCore IV | |
| Memoria (SDRAM) | 256 MiB | | 512 MiB |
| Puertos USB 2.0 | 1 | | 2 |
| Entradas de vídeo | | MIPI CSI | |
| Salidas de Vídeo | | RCA, HDMI, DSI | |
| Salidas de audio | | Conector de 3.5 mm, HDMI | |
| Almacenamiento integrado | | SD / MMC / SDIO | |
| Conectividad de red | ninguna | | 10/100 Ethernet |
| Periféricos de bajo nivel | | 8xGPIO, SPI, I2C, UART | |
| Reloj en tiempo real | | ninguno | |
| Consumo energético | 500 mA, 2.5 W | | 700 mA, 3.5 W |
| Fuente de alimentación | | 5 V Micro USB o GPIO header | |
| Dimensiones | | 85.60mm x 53.98mm | |

Cuadro 2.2: Comparativa entre modelos A y B [2]

### 2.1.2. Raspberry Pi GPIO

GPIO es el acrónimo de General Purpose Input/Output pins[3]. Posiblemente uno de los motivos de mayor éxito de la `Raspberry Pi`. Esta incluye veintiséis pines de los cuales diecisiete son GPIO. Los pines son el medio por el cual la `Raspberry` puede conectarse al exterior.

Los pines son programables por lo que pueden ser usados en multitud de escenarios, ya sea como interruptor de circuitos o como medio de transmisión y/o recepción de datos.

No nos limitemos a pensar en los GPIO como simples interruptores, es cierto que si se diseña

[3]Pines de Propósito General de Entrada/Salida.

y ejecuta correctamente podremos realizar interacciones digitales a base de abrir y cerrar puertas. Pero también disponemos de la capacidad para comunicarnos con verdaderos protocolos de comunicación ya establecidos como son SPI, I2C y UART.

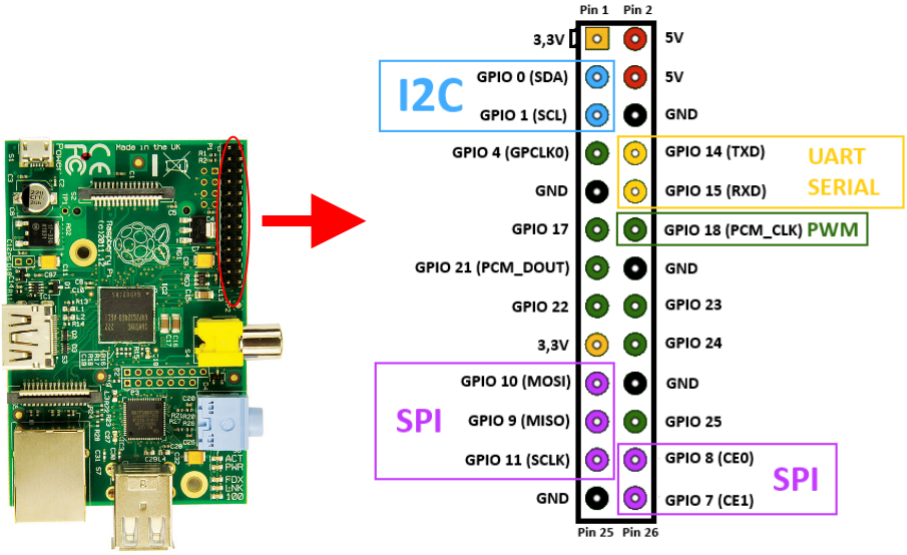



Figura 2.2: Raspberry Pi GPIO [3]

- **SPI:** protocolo de comunicaciones usado para comunicar circuitos integrados. Es una comunicación síncrona que requiere de cuatro conexiones para funcionar:

  1. SCLK: Sistem Clock.
  2. MOSI: Master Output Slave Input.
  3. MISO: Master Input Slave Output.
  4. SS: Slave Select.

- **I2C:** son las siglas de Inter-Integrated Circuit. Este protocolo también se utiliza para comunicar circuitos, aunque generalmente cuando estos se encuentran dentro de la misma placa. Fue desarrollado por 'Philips' en el años 1992. Este protocolo alcanza velocidades de 100 Kb/s en su primera versión y de hasta 3,4 Mb/s en su segunda versión. Aunque normalmente se obvie la última conexión, por ser común, este protocolo requiere tres conexiones:

  1. SDA: datos.

2. SCL: reloj.
3. GND: masa.

- **UART:** Universal asynchronous receiver/transmitter. Se encarga de traducir datos entre serie y paralelo y viceversa.

### 2.1.3. Raspberry Pi Software

Al ser un ordenador de pleno derecho, podemos instalar en el multitud de sistemas operativos distintos y con ellos todo un abanico de software.

Existen actualmente muchas distribuciones compatibles con la `Raspberry Pi`. Ya sean de carácter general o distribuciones específicas. Algunas interesantes son:

- **Raspbian**: Posiblemente el sistema más usado en la `Raspberry`. Se basa en Debian aunque ha sido optimizado para funcionar en la placa `Raspberry`. Desde su lanzamiento en Junio de 2012, ya contaba con 35000 paquetes compilados. Seis meses después aparece la 'Pi Store', tienda que nos permite adquirir contenido, tanto de pago como gratuito, para mejorar nuestra experiencia [5].

- **arkOS**: Creado por Jacob Cook de tan solo 23 años. Este sistema basado en Arch Linux nos permitirá gozar de toda una nube privada en nuestra `Raspberry Pi`. Incluye funciones tales como servidor de correo electrónico, servidor web, bases de datos, etc. Como curiosidad, este proyecto nace cuando su creador se cansa del monopolio de Google y la inseguridad que últimamente genera la NSA (Agencia Nacional de Seguridad estadounidense) [6].

- **OpenELEC**: Orientado a multimedia, este sistema nos permitirá transformar la `Raspberry Pi` en un XBMC (X-Box Media Center) media center. De fácil instalación, cuenta con los códecs necesarios para reproducir casi cualquier archivo multimedia que nos imaginemos. Su éxito se debe a lo bien que aprovecha los recursos del sistema y a su sorprendente velocidad de respuesta. Como detalle, cabe destacar que algunos usuarios han conseguido instalarlo en otros dispositivos como la Apple Tv. Sorprende que ya haya usuarios que prefieran usar este software libre antes que el original del producto [7].

- **RasPBX**: Basado en una imagen Raspbian, este S.O. prescinde de todo elemento gráfico en pro de una gran cantidad de software específico para convertir una `Raspberry Pi` en toda una centralita telefónica (PBX).

  Por motivos ajenos a este proyecto, se realizaron pruebas de telefonía VoIP con este sistema. Los resultados fueron muy prometedores, la `Raspberry Pi` con un sistema RasPBX fue capaz de alojar hasta quince llamadas simultáneas con el códec G729 [8]. Estos datos son sorprendentes, pues es un códec con poca carga en cuanto al ancho de banda pero muy pesado en cuanto a la carga del CPU. Por ello, no hay ninguna duda, que estamos ante una solución telefónica para pequeñas y medianas oficinas [9].

### 2.1.4. Proyectos de la Comunidad

Desde antes de su lanzamiento la `Raspberry Pi` fue víctima de su propio éxito. Dado su reducido coste (alrededor de 35€), lo curioso del proyecto y el entusiasmo de la comunidad, la máquina no tardó en estar agotada. Mientras que los futuros usuarios tuvieron que esperar hasta seis meses a la siguiente remesa, los primeros dueños no tardaron en inundar la red con cientos de proyectos que implicaban como no su nueva `Raspberry Pi`. Así, algunos de los proyectos que más me han llamado la atención son:

- **Supercomputador `Raspberry Pi`**: Uniendo sesenta y cuatro unidades un equipo de ingenieros de la Universidad de Southampton creó un ordenador con una increíble capacidad de carga computacional. Esta máquina fue bautizado como 'Iridis PI'.

  Como con casi cualquier supercomputador que se diseñe, una de las primeras pruebas realizadas fue el cálculo de tantos dígitos de Pi como fuera posible en un tiempo establecido. No se han encontrado los resultados de dichas pruebas, pero si un informe muy bien detallado de las capacidades de este sistema [10].

  Al igual que cualquier buen proyecto que se precie, ofrecen un manual para que podamos montar nuestro propio sistema. Además, no es necesario reunir los casi cuatro mil euros que costó su desarrollo. Con reunir un par de `Raspberry`s ya podremos empe-

zar a realizar desde cálculos sencillos a los más complejos imaginables. Es posible que estemos ante el superordenador menos potente y más barato del siglo.

- **Rpi UAV**: Tras crear el modelo de un dron quad-cóptero, se le añadió una `Raspberry` para controlar las cuatro hélices y así surcar los cielos. Toda la electrónica está controlada por la `Raspberry`: los rotores, los giroscopios, la radio, etc. Este proyecto se sostiene sobre cuatro pilares:

  1. El primero consiste en producir un dron capaz de sobrevolar zonas propensas a incendios. Equipado con sensores de temperatura, gas y cámaras infrarrojas; este dispositivo podría cubrir un gran área y reducir los tiempos de detección de incendios.
  2. El segundo consiste en conseguir un dron barato para realizar operaciones de SAR (búsqueda y salvamento).
  3. El siguiente pretende dotarle de la capacidad de sobrevolar cualquier entorno de forma autónoma, teniendo este la capacidad de volver a base ante cualquier eventualidad.
  4. El último pilar del proyecto consiste en proporcionar los cimientos para otro helicóptero/quad-cóptero de open-source para la comunidad [12].

- **Monitorización del Tráfico**: El equipo de 'Intergreen' ha desarrollado un curioso ingenio para monitorizar el tráfico peatonal y rodado de distintos puntos de la ciudad italiana de Bolzano. Este aparato se compone de una `Raspberry`, unos módulos de Bluetooth y unas baterías; todo ello encapsulado en una caja estanca.

  El proyecto consiste en realizar mediciones del número de dispositivos Bluetooth alrededor de puntos estratégicos de la ciudad. Así podremos encontrarlos en carreteras, calzadas y zonas peatonales. Mediante la monitorización de los dispositivos Bluetooth de su alrededor puede hacerse una idea del numero de dispositivos reales así como de sus velocidades. La tecnología Bluetooth, a pesar de sus deficiencias, fue elegida por su reducido coste de producción y funcionamiento [13].

- **Pi In the Sky**: Seis metros. Eso es la distancia a la que se quedó Dave Akerman de elevar su `Raspberry Pi` 40 Km sobre la línea del mar. Consiguió la increíble cifra de 39,994

metros con unas condiciones extremas: menos de cincuenta grados bajo cero, falta de oxígeno y humedad...

Figura 2.3: Foto tomada desde el aire [14].

La ascensión se realizó mediante un pequeño globo aerostático. Gracias a la cámara instalada de pudieron tomar imágenes como la anterior 2.3. Según cuenta el autor en su blog, la `Raspberry Pi` ofrecía lo que el buscaba: puertos USB para conectar una web-cam y suficiente capacidad de cálculo para manejar el GPS, los datos de los sensores y la radio [14].

## 2.2. Software Empleado

A continuación se hará una breve descripción del software empleado para la realización del proyecto así como de esta misma memoria.

### 2.2.1. Python

Python es el lenguaje de programación elegido para elaborar el proyecto. Su sencilla sintaxis así como su elevado número de bibliotecas hacen que programar sea una tarea sencilla para el usuario. Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis muy limpia y que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License, que es compatible con la Licencia pública general

de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

### 2.2.2. Django

Django es un entorno de desarrollo web escrito en Python que fomenta el desarrollo rápido y el diseño limpio y pragmático.

Django es un framework web de código abierto escrito en Python que permite construir aplicaciones web más rápido y con menos código.

Django fue inicialmente desarrollado para gestionar aplicaciones web de páginas orientadas a noticias de World Online, más tarde se liberó bajo licencia BSD. `Django` se centra en automatizar todo lo posible y se adhiere al principio DRY (Don't Repeat Yourself) [16].

Otra de las características de `Django` es su forma de estructurar las aplicaciones desarrolladas. El framework permite encapsular las aplicaciones permitiendo su reutilización. Una aplicación `Django` reusable es una aplicación que se puede añadir fácilmente a un proyecto y que ofrece una funcionalidad muy específica. Las aplicaciones reusables deberían centrarse en seguir la filosofía Unix de haz una cosa y hazla bien [17]. Esta característica de se aprovechará en este proyecto.

Es necesario mencionar que `Django` se basa en el modelo vista controlador. Donde los datos y la interfaz de usuario quedan separadas. Además existe un controlador encargado de gestionar los eventos del sistema. Esta arquitectura facilita tanto el desarrollo como el mantenimiento pues separa los módulos del sistema.

### 2.2.3. Bootstrap

En 2011, `Bootstrap` se creó como solución interna para solucionar las inconsistencias en el desarrollo dentro del equipo de ingeniería de Twitter. `Bootstrap` es una colección de varios elementos web personalizables y funciones completamente empaquetado en una sola herramienta. Cuando se diseña una web con `Bootstrap`, los desarrolladores pueden elegir qué

elementos utilizar. Aún más importante, tienen la certeza de saber que los elementos que elijan no generarán conflictos entre ellos. Como si se tratase de un puzle, exceptuando que cada pieza del puzle encaja perfectamente con las otras, sin importar la pieza que elija.

Los elementos personalizables de `Bootstrap` son una combinación de HTML, CSS y JavaScript. Gracias a las bondades del Open Source, `Bootstrap` vive en una mejora continua. Se le han añadido una variedad de funcionalidades tales como responsividad 100 % a dispositivos móviles y una selección amplia de plugins `jQuery`. Algunas de las cosas que nos permite conseguir son:

- Interfaces que funcionen de manera brillante en los navegadores actuales, y correcta en los no tan actuales.
- Un diseño que pueda ser visualizado de forma correcta en distintos dispositivos y a distintas escalas y resoluciones.
- Una mejor integración con tus las bibliotecas que sueles usar habitualmente, como por ejemplo `jQuery`.
- Un diseño sólido basado en herramientas actuales y potentes como LESS o estándares como CSS3/HTML5 [18].

### 2.2.4. LaTeX

LaTeX es un sistema de composición de textos de alta calidad. Es un sistema muy diferente a la gran mayoría de editores de texto convencionales del tipo 'What You See Is What You Get' o 'lo que ves es lo que obtienes'. Se compone de comandos TeX con los que el autor puede elegir el tipo de documento a realizar y no preocuparse tanto por el estilo del mismo. Con ello la publicación de textos de alta calidad se vuelve más sencilla.

### 2.2.5. Graphviz

`Graphviz` es un software de código abierto el cual permite la visualización de información mediante diagramas, gráficas y otros tipos de representación. Para ello analiza archivos de texto plano determinando la relación entre las estructuras definidos en ellos.

Para este proyecto se ha utilizado `PyGraphviz`, una interfaz `Python` que permite añadir las funciones del paquete `Graphviz` a nuestro proyecto python. Con él, se ha recreado las dependencias que presentan los diferentes modelos dentro de la base de datos. Esto se puede ver en la sección 3.3 de esta misma memoria.

## 2.3. Electrónica

Dado que será necesario desarrollar circuitos para dotar de mayor funcionalidad al sistema, es conveniente hacer un repaso de algunos de los múltiple componente que podremos utilizar para ello. Por tanto, a continuación se hará un breve repaso de algunos de los más habituales.

### 2.3.1. Sensores

Una de las piezas claves, dado que queríamos otorgarle cierta autonomía al proyecto. La clave de esta pieza de electrónica reside en su capacidad de transformar datos ambientales en datos eléctricos que podemos manejar. Existen multitud de sensores en el mercado. Ya sea por su comportamiento digital u analógico; o por su naturaleza: luz, humedad, temperatura, presión, etc.

Ya sea en el ámbito doméstico o industrial, podemos encontrar sensores en casi cualquier dispositivo hoy en día. Algunas de las características de los sensores son:

- Rango: dominio de la magnitud de medida. Cuanto mayor sea esta variable tanto mejor el sensor.
- Precisión: error de medida esperado.
- Linealidad: comportamiento ante variaciones en la medida.
- Resolución: variación mínima de la medida detectable.

### 2.3.2. Displays

Un *display* no es más que un ingenio que nos permite representar información de manera visual. Los hay tan sencillos como las matrices de LEDs a sistemas mucho más complejos como televisores HD.

- 7 Segmentos: Uno de los *displays* más comunes. Este utiliza siete líneas formando un ocho que se van iluminando según el carácter que queramos representar (como se ve en la imagen). Existen dos tipos de *displays*: 'ánodo común' y 'cátodo común'; según requieran un bit 1 o 0 para encenderse. Habitualmente, estos dispositivos se emplean junto con codificadores que permiten reducir el número de conexiones.

  Cuando los *displays* se nos presentan como un conjunto de dígitos, es decir, se componen de varios *displays* superpuestos, se valen de un efecto óptico para funcionar, pues solo una de los dígitos permanece encendido a la vez. Jugando con la velocidad de refresco, es decir, con el tiempo que un dígito permanece encendido hasta que se apaga y se enciende el siguiente, podremos dotar al *display* de todo un conjunto de efectos.

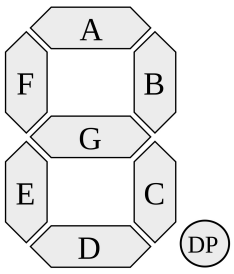



Figura 2.4: Display de 7 segmentos [4].

  Por convenio, cada LED del *display* está indicado con una letra. Esto lo hace más fácil de conectar con otros dispositivos.

- Matriz de LEDS: como su nombre indican, no son más que un conjunto ordenado de LEDS. Cada uno representa un píxel. Por tanto, a mayor número de LEDs mayor resolu-

ción. Para su implementación suelen apoyarse en otros componentes como 'registros de desplazamientos', multiplexores y codificadores.

### 2.3.3. Componentes Eléctricos

Además de los componentes activos y pasivos habituales en los circuitos: generadores de tensión y corriente, resistencias, bobinas, etc. También podemos encontrar otra serie de componentes más específicos. He aquí un breve resumen de algunos de ellos:

- **Relés**: dispositivos electromagnéticos capaces de abrir o cerrar un circuito mediante la aplicación de una señal eléctrica. Es decir, son interruptores accionados por corriente. Permiten además separar de forma independiente dos circuitos incompatibles por sus niveles nominales de tensión y/o corriente.

  Al recibir una señal eléctrica la bobina de su interior genera una campo magnético que hace bascular el electro-imán de su interior abriendo o cerrando el contacto.

- **Motores**: convierten la energía eléctrica en energía mecánica produciendo un movimiento rotatorio. Se componen de espiras e imanes. Cuando circula corriente por dichas espiras estas sufren un par de fuerzas debido al influjo del campo magnético de los imanes.

- **Servomotores**: a diferencia de los motores, los servos no ofrecen un movimiento continuo sino que nos permiten realizar un movimiento de forma controlado. Además de incluir un motor, los servos también incluyen un sensor y una unidad de control. Cuando queramos realizar un movimiento, el sensor calculara la posición o estado del mecanismo e informará a la unidad de control, luego esta determinará cuanta alimentación necesita el motor.

  Son muy habituales encontrarlos en el mundo del aeromodelismo y la robótica.

### 2.3.4. Circuitos Integrados

También conocidos como chips los circuitos integrados son pastillas semiconductoras bajo circuitos en miniatura y encapsuladas para su protección. Esta tecnología nace de la necesidad de simplificar los circuitos a la par que se reducen los costes de producción de los mismos. El primer chip nace de la mano de Jack Kilby en 1959 en los laboratorios de 'Texas Instruments'.

Dado su éxito, fue galardonado en el año 2000 con el premio Nóbel a la Física.

En cuanto a su diseño, existen al menos 3 tipos de circuitos integrados:

1. *Monolíticos*: fabricados de un único cristal, generalmente silicio o germanio.
2. *Híbridos de capa fina*: compuestos de varios materiales, al no poder realizar todos los componentes de un único cristal.
3. *Híbridos de capa gruesa*

En cuanto a su función, existen cientos. Estos son solo algunos:

- **ADC/DAC**: convertidores analógico a digital y digital a analógico. Piezas claves de la electrónica digital. Dado que vivimos en un mundo analógico es necesario tener un mecanismo que nos permita transformar la información de una naturaleza a otra.

  Para poder pasar datos analógicos a digital usaremos un ADC, que realiza los siguientes pasos:

  1. Muestreo: ya que que los sistemas electrónicos funcionan con ciclos de reloj, el tiempo que manejan es discreto. Ello nos fuerza a coger únicamente ciertas muestras de la señal descartando el resto. También es cierto que el Teorema de Muestreo de Nyquist demuestra que si muestreamos a una frecuencia mayor al doble de la frecuencia máxima de una señal continua limitada en banda podremos luego recuperar dicha señal. Como no todas las señales cumplen este criterio, este proceso introduce cierto error en la medición.
  2. Cuantificación: otra de las grandes diferencias entre el mundo analógico y el digital es que el primero trabaja con un rango infinito de valores, mientras que el segundo solo maneja un conjunto finito. El numero de valores posibles viene determinado por el número de bits que maneje el sistema. La cuantificación aproxima los valores de tensión de la muestras a los valores que el sistema maneja. Dado que aquí se están realizando aproximaciones, añadimos cierto error en las medidas cuantificadas. Existen multitud de logaritmos de cuantificación. Cabe destacar dos de ellos

muy conocidos en el sector de las telecomunicaciones. Estos son los algoritmos logarítmicos 'Ley-A' y 'Ley-Mu'. Ambos muy similares entre si. Estos algoritmos se usan para cuantificar la voz, principalmente, en servicios telefónicos. El primero de ellos fue el estándar europeo y el segundo el americano y japonés. Actualmente han sido desplazados por otros códecs.

3. Digitalización: la última parte del proceso consiste en convertir los datos cuantificados en datos binarios. Este proceso es bastante sencillo realmente, dado que solo existen una cantidad de valores cuantificables, la relación es directa.

Para realizar el proceso contrario usaremos un DAC. El procedimiento de reconstrucción de la señal consistirá en:

1. Cuantificación: como se ha descrito antes, el proceso es directo. Para cada conjunto de bits existe una relación directa con el conjunto de valores de tensión disponibles.
2. Interpolación: en este punto nos encontramos ante una señal discreta la cual queremos convertir en continua. Para ello el sistema deberá estimar los valores restantes. Este proceso se realiza mediante interpoladores. Existen infinidad de algoritmos para ello.
3. Filtrado: dado que el proceso de interpolación suele añadir un pequeño margen de error, usamos filtro para depurar la señal.

- **Multiplexores**: circuitos con múltiples entradas y una única salida. Además, tienen la capacidad de seleccionar en cada momento la entrada a la cual quieren dar salida. Se utilizan mucho en el sector de las telecomunicaciones ya que son el puente de unión entre jerarquías de comunicaciones plesiócronas (PDH) y síncronas (SDH). Su antónimo es el demultiplexor. Su función es precisamente la contraria, tiene una entrada y múltiples salidas. Existen miles en el mercado, por citar alguno está el modelo **74HCT238** el cual permite demultiplexar tres puertas en ocho líneas.

- **Codificadores**: son circuitos combinacionales con $2^N$ entradas y N salidas. Existen dos tipos según su prioridad:

1. Codificadores sin prioridad: solamente permite la activación de una entrada en un momento dado. Podrían generar dudas cuando se activasen varias entradas llegando al extremo de a pesar de conocer la salida, desconocer la entrada.
2. Codificadores con prioridad: se consideran entradas más prioritarias que otras. En caso de encontrarse activas dichas entradas eclipsarían la salida correspondiente a las menos prioritarias. Se incluyen dos señales de control para indicar si ninguna entrada está activa o si lo esta la entrada cero.

## 2.4. Proyectos similares

Antes de continuar me gustaría hacer mención a un par de proyectos ya desarrollados que guardan cierta similitud al proyecto que aquí se desarrolla.

- **WebIOPi**: WebIOPi en un framework que permite controlar los pines GPIO de forma local o remota. Incluye las siguientes características:

  1. REST API
  2. HTTP Server
  3. COAP Server
  4. Capa de comunicación
  5. GPIO
  6. Srial/UART
  7. I2C
  8. SPI
  9. Código para más de 30 dispositivos
  10. Bibliotecas Python
  11. Clientes JavaScript

  Como se puede comprobar es una plataforma muy completa que junto a un buen entorno nos permitirá manipular cualquiera de los dispositivos más habituales, gracias a las bibliotecas que incluye, así como cualquier otro circuito que queramos conectar pues también nos permite usar nuestros propios *scripts* [19].

- **Web Control of `Raspberry Pi` GPIO**: Este programa es algo más modesto que el anterior, pero demuestra el entusiasmo que generan proyectos de esta índole.

  Gracias a este otro *framework* podremos conectar algunos circuitos a nuestros GPIO y manipularlos desde la interfaz. No cuenta con las bibliotecas de WebIOPi por lo que se

limita a alimentar circuitos, no existe comunicación entre ellos y la `Raspberry`. Sin embargo estamos hablando de una aplicación sencilla que con algo de ingenio nos brindará la posibilidad controlar una buena variedad de elementos, lo que nos permitirá montar un sistema domótico en nuestros hogares [20].

# Capítulo 3

# Implementación y Desarrollo del Proyecto

## 3.1. Objetivo del Proyecto

El objetivo del proyecto es crear una plataforma web que permita al usuario manipular los distintos componentes electrónicos que se conecten a los conectores GPIO de la `Raspberry Pi`. Esta plataforma además permitirá crear entornos en los cuales el funcionamiento de algunos componentes dependerá directamente de otros.

Una vez instalados el software y la electrónica seremos capaces de tener un elevado grado de control sobre los mismos. Además también seremos capaces de crear 'reglas' que permitirán a la `Raspberry` tener ese control por nosotros. Es decir, podremos darle cierta autonomía.

## 3.2. Estructura del Proyecto

Para su desarrollo fue necesario realizar una división de los posibles tipos de elementos que pudiesen conectarse en un momento dado. También era necesario desarrollar los *scripts* que permitiesen la comunicación entre la `Raspberry Pi` y la electrónica.

Dado la gran diversidad de componentes electrónicos disponibles, se decidió limitar el desarrollo del proyecto únicamente a sensores y circuitos. Así, finalmente, esta división quedó más o menos así:

- Sensores: ya hemos mencionado esta categoría suficientes veces en esta memoria. Son

capaces de generar información en función del entorno.

- Circuitos: elementos que no requieren más que electricidad para funcionar. La `Raspberry Pi` actuará como pila e interruptor.
- Sistemas: conjunto de sensores y/o circuitos. Esta herramienta permite monitorizar un subconjunto de elementos seleccionados. También permite programar el funcionamiento de los circuitos en función de los valores de los sensores.

Por cada una de las categorías anteriores se creó una aplicación `Django`, del mismo nombre, que dota al proyecto del software necesario para manejarlas. También se creo una cuarta aplicación llamada 'raspberry' que almacena algunos parámetros necesarios para el proyecto.

Al descartar la idea de crear una plataforma para conectar cualquier dispositivo y centrarnos únicamente en sensores y circuitos, se decidió realizar el proyecto de forma modular. Es decir, cada aplicación `Django` es lo más independiente posible de las demás. De esta forma se podría desarrollar una nueva aplicación para manejar por ejemplo *displays* sin tener que modificar el resto del proyecto.

Por tanto cada aplicación desarrollada, salvo *sistemas*, está autocontenida[1]. Es decir, cada una contiene toda la información necesaria para poder funcionar dentro del *framework* `Django`. Estas contienen sus propios *scripts*, plantillas HTML, modelos para la base de datos, etc.

### 3.2.1. Aplicación 'Sensores'

Se trata del primer módulo del proyecto. Mediante este módulos podremos añadir, manipular, monitorizar y eliminar sensores.

Para empezar es necesario añadir un sensor. Esto se hace a través de un formulario en la URL: */sensores/add_sensor/*. En el mismo, se han de especificar una serie de características: nombre, GPIO pin y naturaleza de la medida (luz, temperatura, humedad).

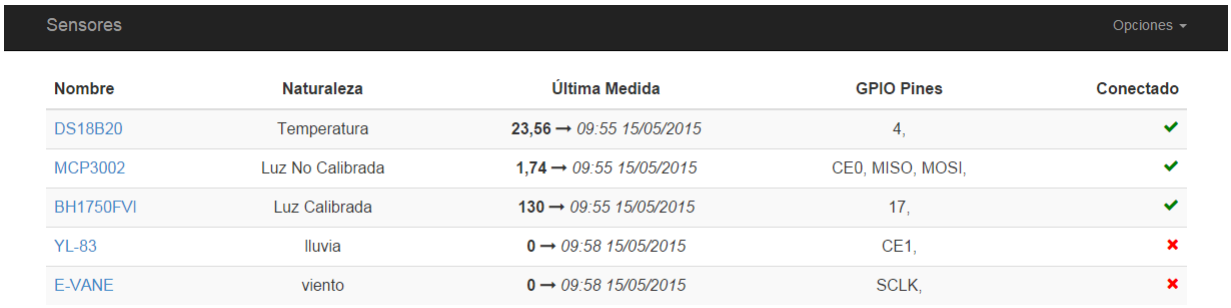
[1] Con la única excepción de depender de la aplicación 'raspberry'

Una vez creado ya podremos empezar a trabajar con el. Para poder manipular los sensores así como comprobar su estado y sus medidas, se ha desarrollado un entorno gráfico simple a la par que amigable, como podemos comprobar accediendo a las siguientes URLs:

/space/sensores/lista/

/space/sensores/*nombre_sensor*/resumen/

Si accedemos a la primera URL podremos ver información del conjunto de los sensores almacenados en la base de datos. Como podemos ver en la figura 3.1, se aprecian los pines GPIO usados, las últimas medidas tomadas y si los sensores se encuentran o no conectados a la `Raspberry Pi`.




| Nombre | Naturaleza | Última Medida | GPIO Pines | Conectado |
|---|---|---|---|---|
| DS18B20 | Temperatura | **23,56** → *09:55 15/05/2015* | 4, | ✔ |
| MCP3002 | Luz No Calibrada | **1,74** → *09:55 15/05/2015* | CE0, MISO, MOSI, | ✔ |
| BH1750FVI | Luz Calibrada | **130** → *09:55 15/05/2015* | 17, | ✔ |
| YL-83 | lluvia | **0** → *09:58 15/05/2015* | CE1, | ✖ |
| E-VANE | viento | **0** → *09:58 15/05/2015* | SCLK, | ✖ |

Figura 3.1: Captura del entorno de la lista de sensores.

Accediendo a la segunda URL se nos redirigirá a la interfaz de un sensor en concreto. Como se puede comprobar en la figura 3.2 este interfaz nos presenta una ventana con toda la información y herramientas disponibles. Además se ha incluido una gráfica que permite visualizar los datos registrados con gran comodidad. Desde aquí también podremos acceder a los ajustes del sensor y modificar algunos de sus parámetros.

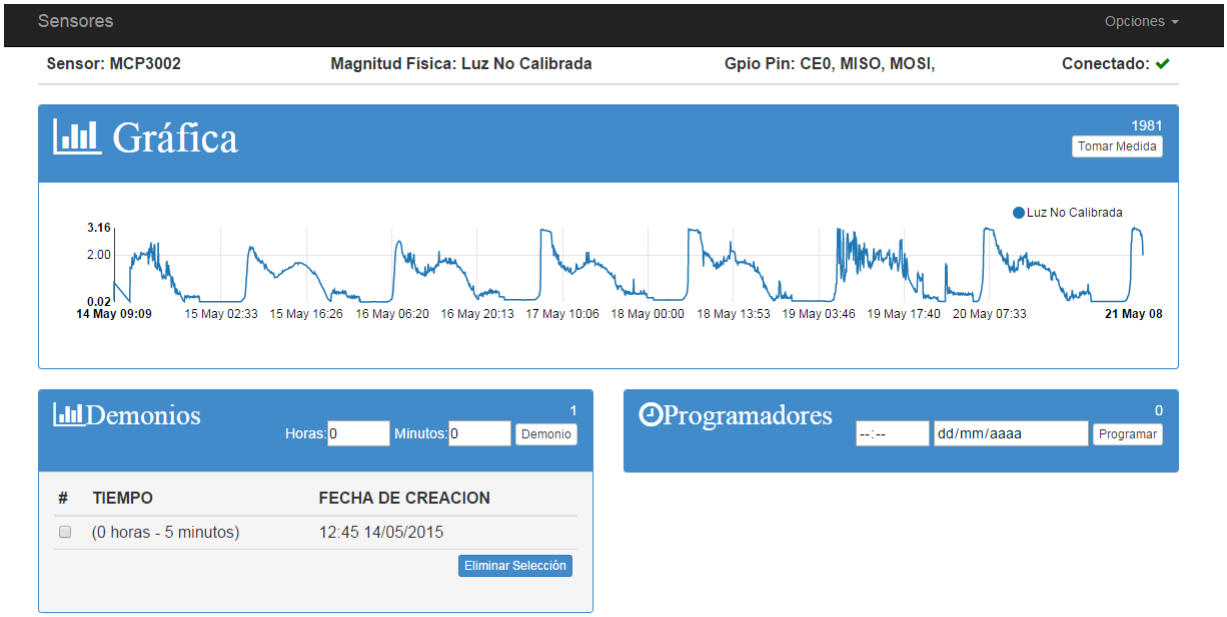



Figura 3.2: Captura del entorno de la aplicación ‘sensores’.

Así, una vez instalado un sensor podremos empezar a obtener datos del mismo. Para ello se han programado tres herramientas que pueden llegar a ser de utilidad para el usuario:

- **Tomar Medida**: con un simple comando la `Raspberry Pi` tomará una única medida que inmediatamente será almacenada en la base de datos y podrá ser visualizada en la gráfica.

- **Programar Medida**: a diferencia del caso anterior, para llevar acabo esta acción deberemos especificar una fecha y una hora. Llegados a ese momento la `Raspberry Pi` , al igual que en el caso anterior, tomará una medida y la almacenará en la base de datos.

- **Demonizar Medidas**: la última herramienta implementada consiste en un demonio (programa en segundo plano). Gracias a este podremos especificar un intervalo de tiempo (horas y minutos) y realizar medidas separadas por el mismo intervalo. Por ejemplo, podremos realizar medidas cada cinco, diecisiete o cuarenta minutos; así mismo, también podremos especificar intervalos del tipo X horas e Y minutos. Para el desarrollo de esta herramienta se ha utilizado ‘cron’ por lo que es el sistema operativo el que controla la activación de los mismos.

Dado que la `Raspberry Pi` no incluye ningún ADC, todos los sensores utilizados han de ser digitales o han de estar conectados a un ADC externo. Esto implica que la comunicación

entre los sensores o los ADC es digital y, por tanto, tendrá que usar uno de los protocolos de comunicación compatible. Esto dificulta mucho el desarrollo de la aplicación pues implica que cada sensor requerirá un *script* específico para funcionar. La única opción posible para el usuario es desarrollar dicho *script* y colocarlo en la carpeta habilitado para ello:

*/sensores/scripts/nombre_sensor/nombre_sensor.py*[2]

Otra solución que se baraja para este inconveniente es permitir al usuario adjuntar dicho *script* cuando rellene el formulario para añadir el sensor. Así, sería el sistema el que automáticamente colocase el *script* en la carpeta correspondiente. Sea como sea, es responsabilidad del usuario usar un *script* válido, compilable y libre de errores.

Como ya se ha comentado en diversas ocasiones, los sensores representan una parte muy importante del proyecto, pues son los encargados de recopilar la información que determinará el comportamiento de otros elementos conectados.

### 3.2.2. Aplicación 'Circuitos'

Esta representa la segunda parte del proyecto. Como se comprobará, esta aplicación guarda gran similitud con la anterior en cuanto a su implementación. Sin embargo, aquí estamos tratando con simples circuitos eléctricos. Como podemos suponer, esta parte ha sido en principio más sencilla ya no requería de *scripts* específicos para cada circuito como si ocurría con los sensores. En esta ocasión se escribió un *script*, que usan todos los circuitos, que cambia el estado del GPIO pin que se le pase como argumento y lo mantiene en ese estado.

Para empezar con esta parte se desarrolló un modelo de 'circuito'. Dicho modelo incluye cuatro parámetros: un nombre identificativo, un GPIO pin, su estado (encendido o apagado) y una entrada lógica (True, False) que indica si esta conectado o no a la `Raspberry Pi`.

Si lo analizamos bien, comprobaremos que lo que estamos implementando es un interruptor que abre o cierra la corriente hacia un circuito eléctrico.

[2]Representa una dirección del S.O de la `Raspberry Pi` no a una URL.

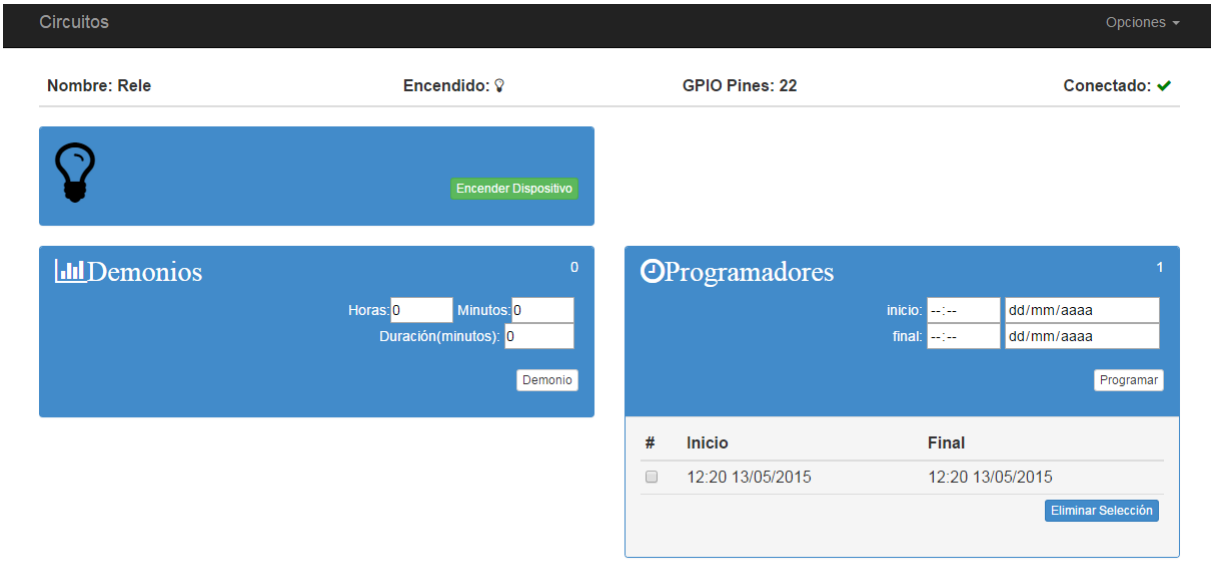



Figura 3.3: Captura del entorno de la aplicación ‘circuitos’.

Tras comprobar el funcionamiento del *script* y la correcta variación del estado del circuito al recibir las señales de apagado y encendido; se comenzó ha desarrollar las herramientas que facilitarán su uso:

- **Encender/Apagar**: nos permite encender y apagar el circuito en cuestión con una simple orden. Sería el equivalente a pulsar el interruptor de casa. Quiero dejar claro que esto no es un pulsador que cambia de estado a cada pulsación es un interruptor en toda regla con dos posiciones diferenciadas: una para encender y otra para apagar. Cada petición indica expresamente el estado final que se quiere dar al circuito. Aunque ambas peticiones hagan un ‘POST’ a la misma URL:

  /interruptor/*nombre_del_circuito*/encender

- **Programar Encendido**: al igual que ocurría con los sensores, esta herramienta nos permite establecer la hora y fecha de un encendido del circuito. Sin embargo, esta herramienta además, nos solicita que establezcamos también la hora y fecha de apagado. Así, este modo escribe dos entradas en la tabla ‘cron’. Una con la orden de encendido y otra con la de apagado. Esta herramienta nos permite programar fechas pasadas (por lo que nunca llegarán a ejecutarse) para así jugar con el programado únicamente del encendido o apagado.

- **Demonizar Encendido**: igual que en el caso anterior. Se nos solicitará un espacio temporal compuesto por horas y minutos que determinarán el periodo tras el cual se dará la orden de encender el circuito, por otro lado, también se nos pedirá la duración, expresada en minutos, que queremos mantener el circuito encendido. Es importante recalcar que el sistema no nos permitirá combinaciones que no cumplan:

$$DURACIÓN < HORAS \times (60) + MINUTOS$$

Las dos últimas herramientas se implementan mediante entradas en la tabla cron. Por lo que ya, desde estos momentos, estamos empezando a dotar a la `Raspberry` de cierta autonomía al darle el control de los circuitos a ella.

Esta aplicación es algo más abstracta que la anterior. Cuando tenemos los sensores sabemos lo que queremos: recopilar información; es trivial. Sin embargo, con la aplicación 'circuitos' lo que se ofrece es una plataforma ¿para encender o apagar qué? Lo que queramos. En las últimas secciones del 'Estado del Arte' se ha intentado concienciar un poco al lector sobre este aspecto. Más adelante, incluso, se presentaran algunos proyectos que se pueden realizar con las herramientas presentadas.

### 3.2.3. Aplicación 'Sistemas'

Gracias a esta última aplicación conseguimos dotar de total automatismo a nuestra `Raspberry`. Mediante una serie de *reglas o condiciones* podremos programar la `Raspberry Pi` para controlar los circuitos conectados en función de los sensores.

Con esta herramienta podremos unificar los elementos instalados en las anteriores aplicaciones y permitirles trabajar como un único sistema. La `Raspberry Pi` se convierte en el núcleo de todos estos elementos encendiendo y apagando los circuitos en función de las mediciones de los sensores.

El primer paso consiste en la creación de un sistema. Para ello rellenaremos el formulario que encontraremos en la URL: /sistemas/add_aplicacion/. Entre otros detalles, nos solicitará los

sensores y circuitos que queramos incluir. Una vez creado se nos redirigirá a lo que podríamos calificar como el panel de control del sistema (véase la figura 3.4). Desde aquí podremos ver el estado de todos los elementos del sistema así como las normas que marcan el funcionamiento de los mismos.


Figura 3.4: Captura del panel de la aplicación 'sistemas'.

Esta aplicación, como ya habrá descubierto el lector, depende de las aplicaciones anteriores: sensores y circuitos; y por tanto es indispensable tenerlas instaladas previamente. Este requisito se da por dependencias en la base de datos, como se muestra en la figura 3.5. Y por dependencia entre *scripts*, ya que como veremos ahora esta aplicación se aprovecha de las tareas programables y de segundo plano que proporcionan las otras aplicaciones.

El siguiente paso tras la creación de un sistema es la creación de 'tareas'. Estas nos permiten programar el encendido u apagado de un determinado circuito[3]. Además, estas tareas pueden estar condicionadas por los valores adquiridos por uno o dos sensores. Las condiciones impuestas a los sensores pueden ser del tipo 'media' donde se realizara la media de las ultimas veinticuatro horas o del tipo 'ultimo valor' donde solo se tendrá en cuanta el último valor recogido. Por

[3]A diferencia de la aplicación circuito donde programábamos el encendido y el apagado o la duración del mismo; aquí solo podemos escoger una de las opciones. Por tanto, si quisiéramos programar el encendido y el apagado, tendríamos que crear dos tareas.

tanto, existen cuatro tipos de tareas distintas:

1. **programador circuito**: usa las herramientas ofrecidas por la aplicación 'circuitos' para programar el encendido u apagado del circuito seleccionado.

2. **programador sensor**: en este caso usamos las herramientas ofrecidas por ambas aplicaciones 'circuitos' y 'sensores'. Dependiendo del tipo de modalidad de condición seleccionada: media o última; se creará un demonio (cada cinco minutos) o un programador (a la hora seleccionada) del sensor. Luego programaremos otro *script* que compruebe si se cumple la condición y actúe en consecuencia con el circuito.

3. **demonio circuito**: en este caso creamos un demonio del circuito para que se encienda o apague periódicamente.

4. **demonio sensor**: en este caso solo se usan demonios. En caso de seleccionar la media como condición, se creará un demonio del sensor cada cinco minutos. Sin embargo, si se escoge como condición 'última medida' se creara un demonio del sensor que se ejecute con la misma periodicidad que el *script* que comprueba las condiciones.

Para realizar las tareas de los circuitos se recurre exclusivamente a las herramientas ofrecidas por dicha aplicación. Sin embargo, para usar las otras dos tareas se recurre también a las herramientas de los sensores. Además ha sido necesario el desarrollo de un *script* que compruebe el cumplimiento de dichas condiciones. Este *script* es ejecutado a la hora programada y recorre las condiciones de una misma tarea, y de cumplirse todas, modifica el estado del circuito correspondiente.

Pero, ¿qué es exactamente una condición? No es más que una entrada en la base de datos que relaciona un circuito con un sensor y un valor. Cuando creemos una condición se nos solicitará:

1. Sensor: sensor al que corresponderán las medidas.

2. Valor: umbral a partir del cual consideramos la variación de estado del circuito.

3. Modo: Media o Último. Indican si queremos comprobar la media de las últimas veinticuatro horas o solo la última medida realizada.

4. Condición: Mayor, Igual o Menor. Con esta última opción indicamos al sistema la relación que esperamos que haya entre entre el valor medido y el valor indicado.

Una vez creada las tareas, será función de la `Raspberry` manejar los componentes. Nosotros por nuestra parte solo nos queda aprovechar estas funciones para crear sistemas tan complejos como se nos pueda ocurrir.

## 3.3. Base de Datos

Cada una de las aplicaciones `Django` mencionadas anteriormente requiere alojar información para su funcionamiento. Para ello nos apoyamos en las herramientas facilitadas por el mismo *framework* `Django`. Solo tenemos que definir la estructura de datos en los archivos *models.py* de cada aplicación, posteriormente `Django` se encargará de generar las tablas pertinentes dentro de la base de datos. Para este proyecto se ha escogido una base de datos `SQLite3`.

Para poder ilustrar al lector con una representación de la estructura de datos implementada se ha recurrido al software `Graphviz`, el cual permite crear gráficas de los modelos programados en `Django`.

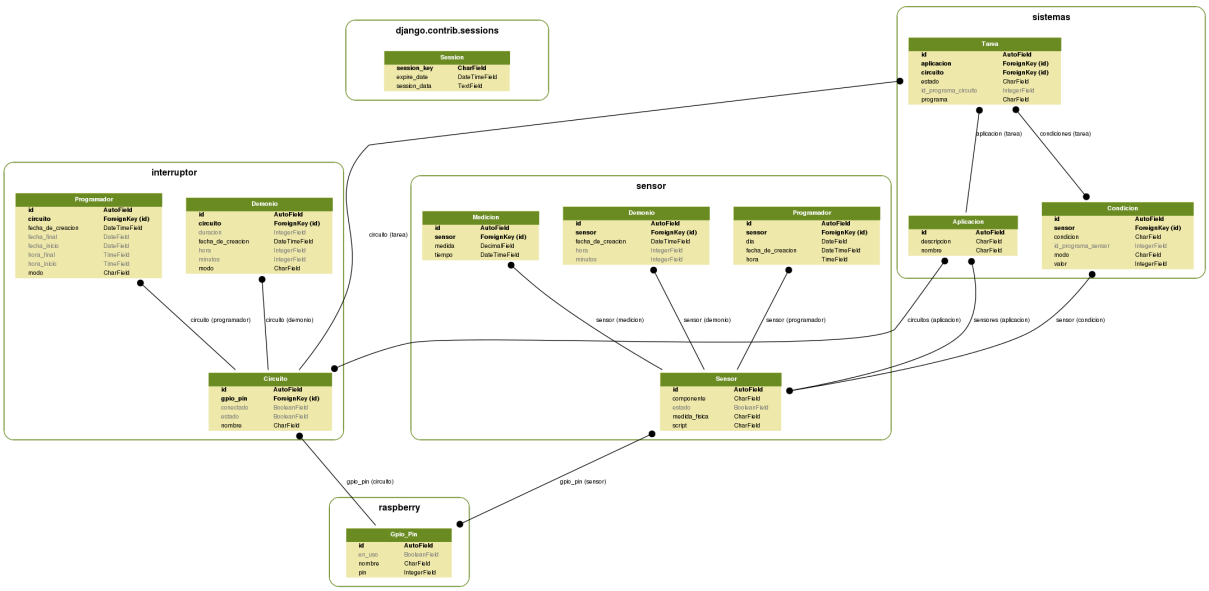



Figura 3.5: Estructura de datos de la base de datos.

En la figura 3.5 podemos ver la estructura de los modelos. En ella se aprecia como se agrupan los modelos en sus respectivas aplicaciones y la fuerte relación que existe entre ellos, tanto a nivel local (aplicación) como a nivel global (proyecto). En total se han desarrollado once modelos agrupados en cuatro aplicaciones para conformar el proyecto y para poder desarrollar todas las herramientas mencionadas hasta el momento. Cada aplicación consta de un modelo base del cual se van ramificando sus correspondientes herramientas.

## 3.4. Desarrollo del Proyecto

A continuación se hará un repaso de la estructura del proyecto.
Tal y como se comentó, al principio de la memoria, antes de comenzar a trabajar con `Django`, era necesario trabajar con la electrónica y con el software, el cual permite su funcionamiento junto con la `Raspberry`. Por tanto, fueron los *scripts* de los distintos componentes lo primero en ser desarrollados. Algunos de estos son:

- *sensores digitales*: para cada sensor digital conectado a la `Raspberry` necesitamos un *script* específico. Ello se debe a que no todos los dispositivos usan los mismos protocolos de comunicación ni las mismas codificaciones.

- *sensores analógicos*: al no contar la `Raspberry` con su propio *ADC*, fue necesario instalar uno. Se adquirió el circuito integrado **MCP3002** el cual permite seleccionar entre dos entradas independientes. Funciona a una frecuencia entre [75-200]KHz y tiene una resolución de diez bits. Para comunicarse con la `Raspberry Pi` utiliza el protocolo de comunicación *SPI*. Para su desarrollo fue necesario recurrir a la biblioteca *spidev* la cual proporciona la base para el protocolo SPI.

- *script para circuitos*: un *script* muy sencillo. Su finalidad consiste en abrir y cerrar un puerto GPIO. Para su desarrollo se optó por crear un objeto (programación orientada a objetos) que se definiese por un único atributo: el puerto GPIO; y dos métodos que permiten la apertura y cierre del mismo.

- *script* display: aunque luego no fue incluido en el proyecto, también se realizó un *script* que permite la visualización de datos en un *display* de ocho segmentos y cuatro dígitos. El

código está escrito para funcionar con un *display*, un multiplexor y un codificador con lo que se reducen las conexiones necesarias entre la `Raspberry` y el circuito. También se programó como un objeto. Para crearlo basta con pasarle como argumentos los pines del codificador (determinarán el símbolo a representar) y los del multiplexador (selecciona el dígito por el cual se representará el símbolo).

- *script XBee*: tampoco se acabo incluyendo este módulo en el proyecto. Sin embargo si que se llegó a desarrollar un software que permitiese a la `Raspberry` comunicarse con otros dispositivos mediante los protocolos 802.15.4 y Digimesh[4]. Al final se acabo utilizando la `Raspberry` como un gateway en WSN (redes inalámbricas de sensores).

Una vez recopilados los *scripts*, iniciamos el desarrollo del servidor `Django`. Para crear el entorno de trabajo recurriremos a las herramientas del mismo *framework*. El proyecto recibe el nombre de *'myproject'*. El entorno creado por `Django`, la carpeta myproject, engloba todas las herramientas básicas que hacen posible el funcionamiento del servidor, sin embargo, insuficientes para la completa realización de este trabajo.

El siguiente paso fue desarrollar las distintas aplicaciones que componen el proyecto. Estas se implementaron en el siguiente orden: sensores, circuitos y sistemas. Para ello se volvió a recurrir al *framework* `Django` el cual también es capaz de generar todo el contenido básico de una aplicación. A medida que se iban trabajando las aplicaciones, se iban implementado a la par las herramientas de las mismas. Finalmente se optó por dar un estilo similar a cada aplicación: archivos views.py, models.py y forms.py; y carpetas *scripts* y templates.

- *views.py*: tal y como se mencionó en el apartado 2.2.2, `Django` sigue un paradigma similar al de modelo-vista-controlador. Este archivo representa el controlador de la aplicación. Responde a los eventos del usuario y el propio sistema, generalmente peticiones PUT y GET. Actúa como intermediario entre el modelo y la vista (interfaz de usuario).

- *models.py*: determina la estructura de datos. Representa el modelo del sistema. En este fichero se diseñan las futuras tablas de la base de datos. En la sección 3.3 se explica más detalladamente.

[4]Variante del protocolo 802.15.4 desarrollado por Digi

- *forms.py*: permite la creación de formularios dinámicos. Dada la continua necesidad de introducir datos en el sistema, se hace fundamental el desarrollo de una estructura eficaz para su manejo. Este sistema nos permite generar formularios que se adapten a nuestras necesidades y a las características de las tablas de datos. Además, esta implementación permite la separación de los formularios y el resto de datos.

- *scripts*: reúne los *scripts* necesarios para la aplicación. Aquí encontraremos los *scripts* de la electrónica así como aquellos que requiera la aplicación: demonios y programadores.

- *templates*: aquí se alojan los archivos HTML, es decir las vistas del proyecto. Define la interfaz del usuario. Permite al usuario desenvolverse en un entorno agradable e intuitivo. Cabe mencionar, que algunas plantillas requieren el uso de funciones Javascript.

Para acabar con el proyecto se decidió utilizar un conjunto de bibliotecas que dotasen de estilo al entorno gráfico por el cual se desenvolverá el usuario. El elemento más destacable es `Bootstrap`. Como se comentó en la sección 2.2.3, este incluye una gran selección de plantillas CSS entre otros aspectos. La plantilla elegida fue *Starter Template*. De estilo algo sobria, ofrece una amplia ventana para ofrecer nuestra vista así como un marco superior que nos permite el rápido desplazamiento por la web. También se han incluido elementos que permiten representar gráficamente los datos de las tablas.

- *django-chartit*: permite representar los datos de la base de datos en diferentes formatos de gráficas. Utiliza bibliotecas Javascript para renderizar los datos. Fue la primera aplicación de este estilo que se probo en el proyecto. Sin embargo acabo relegada en pro de otra aplicación más dinámica: nvd3.

- *django-nvd3*: aplicación completa de bibliotecas gráficas. Permite la reutilización de elementos 'D3.js' por lo que ofrece la posibilidad de crear gráficas dinámicas e interactivas en páginas web. Finalmente fue la aplicación seleccionada para representar las mediciones de los sensores.

Son muchas las dependencias necesarias para la instalación de todos estos elementos. Afortunadamente todos incluyen excelentes manuales en sus respectivas webs.

Una vez instaladas todas las piezas del puzle ya contamos con un proyecto completo. Como hemos podido ver, ha sido necesario recrearse en un variado abanico de aplicaciones. Sin embargo ha sido la única solución para desarrollar el trabajo en su conjunto.

## 3.5. Aplicaciones Implementadas

Ha llegado el momento de realizar alguna implementación que saque partido del proyecto realizado. A continuación se exponen algunos de las aplicaciones desarrolladas:

### 3.5.1. LED y Sensor de Temperatura

Aunque pueda parecer irrisoria su incorporación en este proyecto, este circuito representa el 'Hello World' de la electrónica. No es necesario más que un LED y una resistencia para formar este circuito - la `Raspberry` actuará como pila e interruptor.

Además incluiremos un sensor que mida la temperatura. La `Raspberry` periódicamente comprobará el último valor registrado y alternará entre dos LEDs para indicarnos si se ha superado o no un umbral establecido.

Para realizar este sistema solo necesitamos los elementos de la tabla 3.1:

| Componente | Número |
|---|---|
| LED | 2 |
| DS18B20 | 1 |
| Resistencia | 2 |
| Resistencia 4.7 | 1 |

Cuadro 3.1: Componentes

El primer paso consiste en conectar los LEDs a la `Raspberry Pi`. Para ello podemos apoyarnos en la figura 3.6. Conectaremos un LED rojo al GPIO 18 y un LED azul al GPIO 17.

El siguiente paso será crear una instancia de ambos circuitos en la base de datos. Para ello nos dirigiremos a la URL: */interruptor/add_circuito/*. Una vez agregados ambos LEDs deberíamos comprobar que efectivamente cambien de estado al solicitárselo.

Cierto es que este elemento no da mucho juego por si mismo, sin embargo permite romper

el hielo y adentrarnos de lleno en el mundo del DIY de la electrónica. Una vez perdido el miedo a interconectar dispositivos a la `Raspberry` ya solo nos queda su manipulación mediante software.

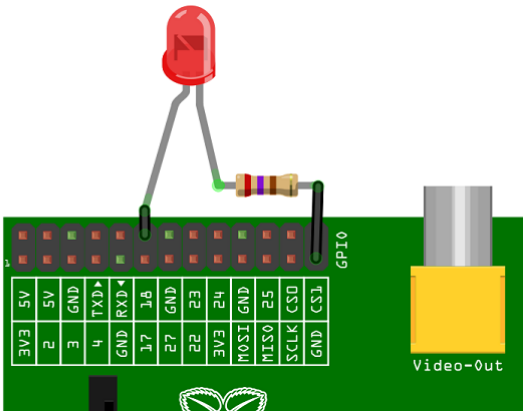



Figura 3.6: Conexión de un LED [22].

Una vez implementado el LED y adquirida la suficiente confianza podremos proseguir. La siguiente pieza a instalar es un sensor. En esta ocasión hemos optado por un sensor de temperatura DS18B20. Este sensor tiene una resolución de doce bits y funciona entre los -55 y +125 grados celsius.

Para instalar el sensor seguiremos los pasos de la sección 3.2.1. Para realizar las conexiones basta con mirar cualquiera de los manuales online. Cuando estemos listo solo debemos navegar a la URL: */sensores/DS18B20/ajustes*[5] y cambiar la su estado por 'activo'.

Una vez instalados todos los componentes pasamos a crear la aplicación y las tareas correspondientes. Para ello seguiremos los pasos de la sección 3.2.3.

Llegados a este punto no nos queda más que dotarle de alguna funcionalidad. En nuestro caso, al realizar una medida superior a 20º C, podemos comprobar como se enciende el LED rojo y apaga el azul, y viceversa si la medida en inferior. Puede parecer una tontería de implementación, sin embargo, ¡estamos ante un sistema muy básico de alerta/alarma!

[5]No es necesario agregar este sensor. Ya existe una instancia del mismo en la base de datos.

### 3.5.2. Riego

Para finalizar el proyecto se ha decidido implementar un sistema real y útil, que mejore el uso de uno de nuestros recursos naturales más importantes, el agua. Para ello, nos apoyaremos en los datos ambientales registrados por una serie de sensores. Esta información permitirá al sistema consumir una cantidad proporcionada de agua.

Para la realización de este sistema me he basado en las condiciones de los dos jardines de mi casa familiar ubicada en Jerez de la Frontera, Cádiz, y en las duras condiciones que azotan la región en los meses más calurosos de año: junio, julio y agosto. Es importante recalcar que ambos jardines son muy similares con la única diferencia de estar ubicados a poniente y levante. Este hecho marcará ciertas diferencias entre las condiciones de riego de ambos jardines.

El primer paso es considerar los efectos ambientales que queremos tener en cuenta para mantener las condiciones de nuestros dos jardines y determinar la duración del riego. Podríamos seguir varias estrategias: usar sensores que midan la humedad de la tierra, que determinen cuando esta está suficientemente húmeda; o podríamos usar sensores de temperatura y en función de la media diaria calcular el tiempo de riego. Sea como sea, vemos que el hecho de añadir sensores nos da mayor grado de libertad. En esta ocasión, los parámetros elegidos son los siguientes:

1. Temperatura: Jerez sufre altas temperaturas durante el verano. Hay días en que las máximas se disparan por lo que es realmente interesante tener controlado este parámetro.

2. Viento: existen dos vientos predominantes en la región, el viento hÃºmedo de poniente y el viento seco de levante. El viento de levante puede resultar demasiado seco y caluroso por lo que se hace indispensable monitorizar el viento en los días cálidos.

3. Lluvia: es muy poco probable que se den lluvias en Jerez en los meses de verano. Sin embargo, de ocurrir, no sería muy práctico volver a regar.

El siguiente paso es determinar el material necesario. Pero, ¿qué hace falta para automatizar el riego? La pieza básica en este proyecto es una válvula hidráulica. Pero, dado que la `Raspberry` no tiene suficiente potencia para activarla es necesario incluir también un relé. En

esta ocasión nos hemos decantado por:[6]

| Componente | Número | Descripción | Precio |
|---|---|---|---|
| SRD-05VDC-SL-C | 2 | Relé: activará las válvulas de riego | 2,78 € |
| DS18B20 | 1 | Sensor de temperatura | 3.39 € |
| SEN-08942 | 1 | Estación meteorológica. | 67.38 € |
| Rain Bird 100-DV | 2 | Válvula de riego | 20,15 € |

Cuadro 3.2: Componentes electrónicos del riego

Una vez adquiridas todos los componentes pasaremos a su montaje. Instalaremos los sensores y el resto de circuitos. Luego, desde la aplicación web los agregaremos y crearemos el sistema correspondiente.

Tras comprobar que los sensores toman medidas y los relés se encienden y apagan es el momento de crear las tareas que automatizarán el riego. Nosotros vamos a crear cuatro normas por jardín para asegurarnos que se monitorizan todos los parámetros mencionados anteriormente y no exista la posibilidad de dejar el riego encendido. Por tanto crearemos estas cuatro tareas:

1. Lluvia: condicionaremos el encendido del relé al sensor de lluvia de la estación meteorológica. Para ello programaremos un demonio que se ejecute todos los días a las 23:00 horas. Comprobará la media del día, y siempre y cuando esta sea menor que un umbral elegido, activará el relé que a su vez activará la válvula de riego.

2. Temperatura: igual que el caso anterior. Esta vez elegiremos el sensor de temperatura y lo programaremos para actuar diez minutos después del encendido. Al igual que antes, si la temperatura media del día no supera un umbral el riego se apagará.

3. Viento: más de lo mismo. Programamos un demonio para que se ejecute todos los días a las 23:15. Este último tendrá en cuenta la dirección del viento. Si la media no es considerada viento de levante el circuito se apagará finalizando el riego.

4. Apagado: finalmente, apagamos el riego de forma automática todos los días a las 23:20.

Finalmente nuestra tabla de tareas y condiciones queda así:

[6]Los precios han sido extraídos de: http://www.dx.com/, http://www.electronicaembajadores.com/, https://www.sparkfun.com/ y http://www.turiego.es/ respectivamente

| Hora | Estado | Descripción | Condiciones |
|---|---|---|---|
| 23:00 | on | Enciende el riego si lluvia $<$lluvia $_{\text{UMBRAL}}$ | • Todos los días. |
| 23:10 | off | Apaga el riego si temperatura $_{\text{Media}}$ $<$Temp $_{\text{UMBRAL}}$ | • Si no llueve |
| 23:15 | off | Apaga el riego si el viento no es de levante | • Si no llueve<br>• Si tmp $>$Temp $_{\text{UMBRAL}}$ |
| 23:20 | off | Apaga el circuito | • Si no llueve<br>• Si tmp $>$Temp $_{\text{UMBRAL}}$<br>• Hay levante |

Cuadro 3.3: Condiciones de riego

Como podemos comprobar en la tabla anterior, el sistema comprobará las condiciones ambientales para determinar la duración del riego. Ahora solo nos queda configurar el otro jardín y determinar los umbrales de cada sensor.

# Capítulo 4

# Conclusiones

Antes de finalizar con la memoria, convendría hacer un repaso del trabajo realizado así como de las impresiones adquiridas durante su desarrollo.

## 4.1. Proyecto

Al comienzo de la memoria se resaltaron los objetivos de este proyecto. Echando la vista atrás podemos comprobar como se han ido completando las metas:

1. *Diseño e implementación de circuitos.*
   Se han realizado todo tipo de circuitos eléctricos y electrónicos. Se ha jugado con sensores, relés, *displays*, zumbadores, etc.

2. *Diseño del código de le electrónica.*
   Como ya se mencionó anteriormente, era necesario desarrollar un software que permitiera a la `Raspberry Pi` comunicarse o manipular los dispositivos conectados. Así, a medida que se iba añadiendo un componente se iba realizando dicho *script* a la par.

3. *Diseño del servidor* `Django`.
   Como se ha explicado, se han desarrollado tres aplicaciones `Django`. Cada una proporciona unas herramientas específicas.

4. *Implementar todas las funciones del servidor.*
   Esta parte está finalizada pero no completada. Se han desarrollado varias funciones tales

como programadores, actuadores, etc. Sin embargo existen multitud de acciones más que podrían programarse.

5. *Diseño de entorno* `Bootstrap`.
   Totalmente implementada. De no ser por este aspecto el manejo de la aplicación no sería intuitivo.

Como extra, se ha intentado dar un uso práctico al proyecto realizado. Ello ha sido para darle más vida, así como demostrar al lector la utilidad del mismo. Por ello se ha mencionado y explicado en la sección 3.4 algunas aplicaciones.

Además de valorar los objetivos del proyecto en sí, es necesario hacer, también, una valoración de la utilidad del mismo. Es necesario comunicar que este proyecto jamás se ideó con fines comerciales ni con ningún otro que no fuera el de recrearse en un proyecto de electrónica y software que permitiera, a la par que exigiese, aprovechar los conocimientos adquiridos en los últimos años de universidad. Sin embargo, a medida que se va desarrollando el proyecto uno se va dando cuenta de las muchas posibilidades que puede tener un proyecto de este índole. Reflexionemos un momento. ¿Cuántos sistemas no requieren una constante monitorización a día de hoy? Desde grandes edificaciones que quieran medir su eficiencia energética a agricultores que quieran conocer parámetros de sus cosechas pasando por la persona que quiera instalar una caja negra en su vehículo. Por citar algunos ejemplos.

Al inicio de la memoria se mencionó la posibilidad de realizar un proyecto que asentara las bases de otros proyectos. No creo que haya sido finalmente el caso. Sin embargo, si que se podría aprovechar la experiencia adquirida para desarrollar un nuevo producto. Existen multitud de posibilidades en este aspecto. Existe un producto en particular por el que me he decantado. Me refiero a las 'redes de sensores'. Si analizamos bien el proyecto realizado se asemeja bastante a una 'mota', con la excepción de no incluir un módulo de comunicaciones.

Sea como fuere, si quisiéramos desarrollar un producto comercial quizás deberíamos optar por un microcontrolador más barato que la `Raspberry Pi`. Tengamos en cuenta que luego habría que añadir los módulos, la protección y sobre todo la alimentación.

## 4.2. Aplicación de lo aprendido

Para la realización de este proyecto ha sido necesario aplicar muchos de los conocimientos adquiridos durante los años de estudio universitario. Pero en especial lo aprendido en las siguientes asignaturas:

1. *Electrónica y Sistemas Digitales.*

   En estas dos asignaturas se dota al alumno del conocimiento básicos de componentes y sistemas digitales a la par que se adquiere experiencia en su manipulación.

2. *Servicios y Aplicaciones en Redes de Ordenadores.*

   En esta asignatura se enseña lo básico sobre los servicios y aplicaciones comunes en las redes de ordenadores, y en particular en internet [32]. En ella aprendí a manejar el framework Django que ha permitido realizar en este proyecto la parte del servidor. Así mismo también se enseñan a realizar los `templates` que componen la vista de la aplicación web y a dotarlos de estilo con la programación `CSS`.

3. *Comunicaciones Sectoriales.*

   Esta asignatura se imparte en el últiumo semestre de la carrera. Su finalidad no es enseñar al alumno más conocimientos sino darle una imagen práctica de los conocimientos ya adquiridos. Es decir, aquí se presentan diversas tecnologías que se componen de lo aprendido en el grado hasta el momento. Esta asignatura motivó en gran parte este proyecto.

## 4.3. Lecciones aprendidas

También es necesario hacer mención a lo aprendido durante el desarrollo del proyecto:

1. *Protocolos de Comunicación.*

   Durante el diseño de los circuitos fue necesarios entender como funcionaban los diversos protocolos de comunicación mencionados en la sección 2.1.2.

2. *Desarrollo de un Proyecto.*

   Es importante mencionar que también se ha adquirido experiencia en el desarrollo de un proyecto que auna tantas piezas. Desde su diseño hasta su final implementación se

ha tenido que lidiar con multitud de herramientas de distintas índoles. Por lo que fue necesario una planificacíon previa de los pasos a seguir para completar el proyecto.

3. *Manejo de LaTeX.*
   Para el desarrollo de esta memoria fue fundamental aprender a redactar textos con esta herramienta. Su dificultad radica en el hecho de ser es poco intuitivo. Sin embargo esta dificultad se ve reducida a medida que se va adquiriendo soltura con el mismo.

## 4.4. Trabajos futuros

Una vez finalizado el proyecto es posible plantearse una serie de mejoras o extras que podrían dotar a la `Raspberry Pi` de mayores funciones.

Quizás los componentes que mayor juego pueden darle a este proyecto serían los módulos de comunicación. Dotando al sistema de dichos módulos podríamos tener un sistema que no solo se pueda aplicar en un area pequeña, como eran el termómetro 3.5.1 o el riego 3.5.2, sino en un area de mayor extensión. Así mismo sería interesante añadir la capacidad de mostrar información en displays.

En cuanto a mejoras del software cabe mencionar la posiblidad de añadir más herramientas que permitan al usuario realizar distintas acciones con los circuitos y así mismo añadan más funciones a los sistemas implementados. También podría mejorarse la interfaz.

## 4.5. Raspberry

Llegados a este punto, justo antes de finalizar la memoria, creo necesario realizar ciertas valoraciones sobre la que ha sido nuestra compañera de trabajo durante todo el proyecto, la `Raspberry Pi`.

Antes de nada, hagamos un pequeño repaso de lo que es la `Raspberry Pi`. Este no es más que un ordenador del tamaño de una tarjeta de crédito, con una capacidades sorprendentes de cálculo y una memoria RAM, a mi criterio, algo insuficiente. Goza de varios puertos de en-

trada y salida. Y su característica más reseñable es su precio: 35€. No es de extrañar que con este 'pedigrí' haya conseguido convertirse en uno de los ordenadores más famosos y vendidos del planeta en tan solo un par de años. Dada su gran versatilidad: multitud de S.O, proyectos, juegos, etc; podemos usarla en casi cualquier ámbito o rincón que se nos ocurra. Su límite es nuestra imaginación.

Por tanto debemos concluir que pese a no encontrarnos con un ordenador de gran potencia, estamos ante uno de grandes posibilidades. Es por ello, que como producto de recreo o como base de investigaciones, puede resultar una buena opción. Su enorme versatilidad puede comprobarse en los miles de proyectos en los que es protagonista.

# Bibliografía


[1] Foto Raspberry Pi

http://www.farnellnewark.com.br

[2] Raspberry Pi Hardware

http://es.wikipedia.org/wiki/Raspberry_Pi

[3] Raspberry Gpio

http://diymakers.es/

[4] *Display* de Siete Segmentos

http://wikipedia.com

[5] Web de Raspbian

http://www.raspbian.org/

[6] Web de arkOS

https://arkos.io/

[7] Web de OpenELEC

http://openelec.tv/

[8] Ernesto Dos Santos Afonso. *Ancho de banda utilizado por VoIP*

http://www.3cx.es/ancho-de-banda-voip/

[9] Web de RasPBX

http://www.raspberry-asterisk.org/

[10] Simon J. Cox. *Iridis-pi: a low-cost, compact demonstration cluster*

http://www.southampton.ac.uk/~sjc/raspberrypi/raspberry_pi_iridis_lego_supercomputer_paper_cox_Jun2013.pdf

[11] Web de Raspberry Pi en Southampton

http://www.southampton.ac.uk/~sjc/raspberrypi/

[12] Web de Rpi UAV

https://github.com/cTn-dev/RPi-Phoenix

[13] Tom Nardi. *Integreen Brings Open Source Traffic Monitoring To Italy*

http://www.thepowerbase.com/2012/12/integreen\-brings\-open-source\-traffic\-monitoring\-to\-italy/

[14] Web de Dave Akerman

http://www.daveakerman.com/?p=592

[15] Liz Upton. Pin the sky

http://www.raspberrypi.org/pi\-in\-the\-sky/

[16] Web de Django

http://django.es/

[17] Web de Django España

Django.es:http://django.es/blog/convenciones\-aplicaciones\-reusables-django/

[18] Xavier Du Tertre. *¿Qué es Bootstrap? ? La Historia y el Bombo*

http://www.prestashop.com/blog/es/que\-es\-bootstrap\-la\-historia\-y\-el\-bombo\-parte\-1\-de\-2/

[19] Web de WebIOPi

https://code.google.com/p/webiopi/

[20] Web Control of Raspberry Pi GPIO

http://www.instructables.com/id/Web\-Control\-of\-Raspberry\-Pi-GPIO/all/?lang=es

[21] Demultiplexador 74HCT238

http://www.gme.cz/

[22] Conexión de LED

https://www.raspberrypi.org

[23] Sensor de Temperatura DS18B20

http://blog.bitify.co.uk/

[24] Válvula de Riego. Rain Bird 100-DV

http://www.rainbird.com/

[25] Alberto Lozano

[26] http://es.wikipedia.org/wiki/Raspberry_Pi

[27] http://elinux.org/RPi_Hardware

[28] Paul Scherz y Simon Monk. *Practical Electronics For Inventors*. McGraw Hill, 2013.

[29] Simon Monk. *Programming the Raspberry Pi. Getting Started with Python*. McGraw Hill, 2013.

[30] Maik Schmidt. *Raspberry Pi A Quick-Start Guide*. The Pragmatic Programers, 2012.

[31] Ruth Suehle y Tom Callaway. *Raspberry Pi Hacks*. O´reilly, 2014.

[32] Dr. Jesús M. González Barahona, Dr. Gregorio Robles Martínez *Servicios y Aplicaciones en Redes de Ordenadores Grado en Sistemas de Telecomunicación Programa del curso*