_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISEÑO DE UN SENSOR DE NIVEL DE GAS EN UNA BOMBONA DE BUTANO, CON COMUNICACIÓN INALÁMBRICA, CONSULTA POR INTERNET Y AVISO A MÓVIL

Carlos Aznar Ruiz

Tutor: Héctor García Miquel

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2013-14

Valencia, 5 de julio de 2014

AGRADECIMIENTOS

En primer lugar están quienes más tiempo han pasado a mi lado en estos duros meses de trabajo y de sufrimiento: Albert, Gor, Javier, Lorena, Vicente, Víctor…. ¡VA POR VOSOTROS!

No me puedo olvidar de mis padres, quienes no solo me dieron la idea de este trabajo, si no que me han apoyado y financiado mis estudios y este proyecto.

Por último y no menos importante, debo mencionar a Miguel, por aportarme esos conocimientos sobre todo el mundo de Arduino y ahorrarme mucho dinero.

¡El hombre elige! ¡El esclavo obedece!

Andrew Ryan (Bioshock)

Resumen

En la actualidad ha crecido exponencialmente el número de aplicaciones que permiten una comunicación entre electrodomésticos o sensores de un hogar y los terminales móviles, en los cuales se pueden visualizar datos de interés para los usuarios, o bien controlar dichos electrodomésticos. Esto marcará el camino en la electrónica y en la programación, al cual se quiere añadir un grano de arena para hacer la montaña todavía más grande. En el presente proyecto se desarrollará una red de comunicación entre un sensor, el cual estará diseñado para medir el nivel de líquido restante en un recipiente del cual se conoce su contenido y capacidad máxima, dicho sensor estará conectado mediante WiFi a un router que le proporciona acceso a Internet y permite subir la medida del sensor a un servidor Web y que dicha medida pueda ser accesible desde un terminal Android para su visualización. Este caso se ha concretado con la aplicación para poder medir el nivel de gas de una bombona de butano en un domicilio particular.

Resum

A la actualitat ha crescut molt el nombre de aplicacions que permeten una comunicació entre electrodomèstics o sensors d'un domicili i els terminals mòbils, als quals es poden veure dats d'interès per als usuaris, o controlar els electrodomèstics. Aquest marcarà el camí a la electrònica i la programació, al qual es vol afegir un gran d'arena para fer la muntanya encara més gran. Al present projecte es desenvoluparà una xarxa de comunicació entre el sensor, el qual serà dissenyat per a mesurar el nivell de líquid que falta a un recipient del qual es coneix el seu contenint y capacitat màxima, el sensor estarà connectat mediant WiFi a un router que el proporcionarà accés a Internet y el permetrà pujar la mesura del sensor a un servidor Web y que eixa mesura puga ser accessible amb un terminal Android per a la seua visualització. Aquest cas s'ha concretat amb la aplicació per a mesurar el nivell de gas de una bombona de butà a un domicili particular.

Abstract

Nowadays the numbers of applications that allow the communication between electrical appliances or sensors and the mobile terminals have had a large growth, in which is possible to visualize data of interest for the users, or control these electrical appliances. This will mark the way to be in electronics and programming, and it's wanted to add a grain of sand to make the mountain even taller. In the present project it will be developed a communication network between a sensor, which will be designed to measure the level of remaining liquid in a container of which the maximum capacity and content are known, that sensor will be connect by WiFi to a router that will allow it to be connected to the Internet, so it could upload the measured data to a web server and that data could been show by an Android application to the user. In this case the idea has been focused to be able to measure the remaining level of a butane bottle in a particular house.

Índice

Índice de Figuras

Índice de Tablas

Índice de Fórmulas

Capítulo 1. Introducción

En los últimos años ha florecido y crecido un nuevo campo en las tecnologías de la información y la comunicación: la domótica. Se puede definir domótica como un conjunto de sistemas que son capaces de aportar servicios a un usuario de gestión, control, información y bienestar en su vivienda. Usualmente se compone de redes de sensores conectadas mediante cable o inalámbricamente a un terminal que ofrecerá esos servicios al usuario, hoy en día ese terminal suele ser casi siempre un Smartphone.

Este proyecto se ha enfocado como un sistema que pueda ser capaz de medir el nivel restante de un recipiente, conociendo anteriormente su capacidad máxima y la densidad de su contenido. Esta información de nivel de líquido restante será enviada inalámbricamente al terminal del usuario para que sea capaz de verla representada gráficamente.

El sistema que se ha pensado para lograr este objetivo es la colocación de un sensor de peso mediante galgas extensiométricas en la base del recipiente, con su correspondiente acondicionador de señal para lograr una tensión óptima para su paso a digital a través de un convertidor analógico digital (ADC), después se enviará inalámbricamente mediante WiFi a un servidor Web, el cual estará diseñado para albergar estos datos y dar soporte a que dichos datos sean descargados por una aplicación de Android, la cual hará una representación gráfica al usuario.

De todas las posibles aplicaciones de este sistema, se ha escogido particularizarlo en este proyecto para el caso de una bombona de butano en un domicilio, esta elección viene fundamentada por la importancia para el usuario de conocer el nivel de gas restante en su botella y que no se corte el flujo mientras está realizando tareas tales como: cocinar o ducharse.

Capítulo 2. Objetivos del Trabajo Final de Grado

El objetivo de este proyecto es dar solución a un problema que se presenta en muchos hogares, quienes tienen calefacción por gas butano. A estos usuarios les sería de utilidad el conocer la cantidad de gas restante en su bombona y cuánto de ese gas podría ser usado.

Para resolver este problema se pensó en usar los conocimientos adquiridos durante la carrera en materia de comunicación inalámbrica, sensores de presión, programación de microprocesadores y toda la información y formación adquirida por cuenta propia.

El esquema general que se usará seguirá estas pautas:

- Un sensor de peso que medirá la masa de la bombona de butano, y otro de temperatura para conocer el valor de la temperatura ambiente.
- Una placa Arduino instalada en la base donde está el sensor que leerá la medida y se comunicará con otra placa conectada a la corriente eléctrica.
- Esta segunda placa Arduino subirá a un servidor las medidas realizadas, además de la hora actual.
- El usuario verá la medida representada gráficamente en un terminal Android mediante una aplicación que descargará los valores del servidor, así como la diferencia en horas entre la fecha actual y la fecha de la última actualización de los datos.

En resumen: se pretende diseñar un sistema capaz de mostrar en la pantalla de un terminal Android el porcentaje de gas restante en la bombona del domicilio, a través de la medida de su masa. A parte, también se hará una medida de temperatura, la cual se usará para estimar el porcentaje mínimo de gas que tiene que tener la bombona para su correcto funcionamiento. Para ello se utilizarán dos sistemas de comunicación inalámbrica diferentes: WiFi para comunicar la placa UNO con el servidor, y el estándar 802.15.4 para comunicar ambas placas de Arduino.

También se pretende dotar al sistema de la mayor autonomía posible, debido a que la placa Arduino que controla el sensor de peso estará alimentada por pilas, y se pretende ofrecer al usuario la mejor experiencia posible, ampliando al máximo la duración de dichas pilas.

Capítulo 3. Metodología del Trabajo Fin de Grado

3.1 Consideraciones Técnicas

3.1.1 Bombonas de butano

Se entiende por bombonas de butano aquellos recipientes que contienen dicho gas en forma licuada, listo para ser usado por calefacciones en los hogares que dispongan de este sistema de calefacción instalado.

Al usuario se le presentan en dos recipientes diferentes:

- UD-125 (Acero): 13.9 kg en vacío $\pm$ 0.9 kg. 12.5 kg de gas butano.
- Acero Inoxidable: 5.5 kg en vacío. 12.5 kg de gas butano.

El sistema será diseñado para funcionar con la UD-125, que sigue siendo la más utilizada en nuestro país, pero aun así, al ser la otra de un peso inferior, nos aseguramos de que no dañe el sistema, y si bien es verdad que tendríamos una menor sensibilidad en la medida, aproximadamente también podríamos calcular el porcentaje de gas restante en el recipiente.

En ambos envases, la presión del gas butano en la botella es de 2,137 kPa para 20ºC y el contenido del envase se compone en un 80% de butano, mezclado con otros gases, siendo su densidad final de 0.56 kg/m3.

Todos estos datos han sido extraídos del Real Decreto 919/2006, el cual hace referencia a la norma NTP 209. Ambos documentos se encuentran accesibles en los Anexos.

3.1.2 Galgas Extensiométricas

Es un sensor para medir la deformación, presión, carga, etc., el cual se basa en el efecto piezoresistivo, propiedad de los materiales de cambiar el valor nominal de su resistencia cuando se le somete a esfuerzos y se deforman en dirección de los ejes mecánicos. Suelen fabricarse de aleaciones de diferentes metales o de elementos semiconductores.

Para la realización del proyecto se usarán galgas metálicas, cuyas especificaciones se detallarán posteriormente.

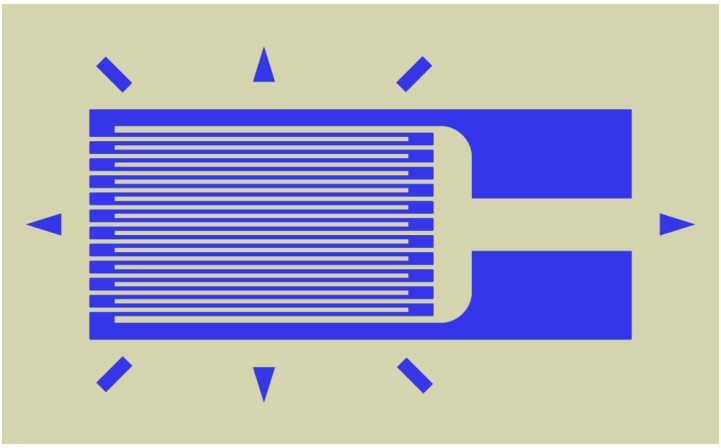


Figura 1. Representación de una galga extensiométrica

Las galgas extensiométricas se basan en la Ley de Hooke, la cual se define como:

$$\sigma = E\,\varepsilon \qquad \textbf{(3.1)}$$

Siendo sigma el esfuerzo al que se somete la galga, que puede ser definido tal que:

$$\sigma = \frac{F}{S} \qquad \textbf{(3.2)}$$

En estas ecuaciones, ε representa la deformación unitaria, $\varepsilon = dl / l$, E es el módulo de Young: coeficiente de alargamiento característico de cada material, esfuerzo que habría que aplicar para obtener un alargamiento igual a la longitud inicial, F sería la fuerza aplicada sobre la superficie S.

La resistencia de una galga puede ser modelada tal que:

$$\mathrm{R} = R_0 + \mathrm{dR} = R_0\,(1 + \frac{dR}{R_0}) = R_0\,(1 + \mathrm{x}) \qquad \textbf{(3.3)}$$

Siendo $x = dR/R0 = k\,\varepsilon$, siendo k el factor de galga, con un valor típico para las galgas metálicas de $k = 2$.

3.1.3 Puente de Wheatstone

Circuito electrónico usado para medir resistencias desconocidas haciendo uso del equilibrio eléctrico de los dos brazos del puente. Cada brazo lo forman dos resistencias, formando un circuito cerrado de cuatro resistencias. El esquema general es el siguiente:

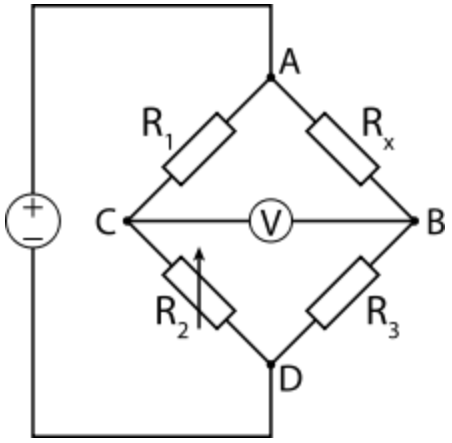


Figura 2. Esquema de un Puente de Wheatstone

Rx será la resistencia que se pretende medir, siendo *R2* un potenciómetro ajustable, cuyo valor se variará hasta alcanzar el punto de equilibrio: *Vc = Vb*, por lo que entre dichos puntos no circulará corriente alguna, quedando la relación de resistencias *R1/R2 = Rx/R3*.

Otro posible método es el cual no se usa un potenciómetro para *R2*, sino una resistencia fija, y conociendo *R1* y *R3*, se mide la corriente que circula de C a B será medida con el fin de conocer el valor de *Rx*.

La aplicación de este diseño para su uso en instrumentación es tan sencilla como sustituir una o varias de las resistencias por sensores, y a través de la diferencia de tensión entre C y B conocer la magnitud física a medir. Debido a que la diferencia de tensiones entre C y B suele ser de valores muy difíciles de medir, mV o uV, se suele utilizar un amplificador diferencial a la salida del puente como acondicionador de la señal para que pueda ser tratada de forma eficiente.

3.1.4 Amplificador de instrumentación con 3 A.O.

El amplificador de instrumentación es un amplificador diferencial tensión-tensión cuya ganancia puede establecerse con gran precisión. Es un elemento indispensable en los sistemas de medida, en los cuales cumple la función de acondicionar la señal medida para aumentar sus características eléctricas para su correcto tratamiento. Se basa en el uso de Amplificadores Operacionales, en este caso, se usarán 3 A.O. Su diseño se caracteriza por ofrecer una alta impedancia de entrada y un alto rechazo al modo común (CMRR). Su estructura es la siguiente:

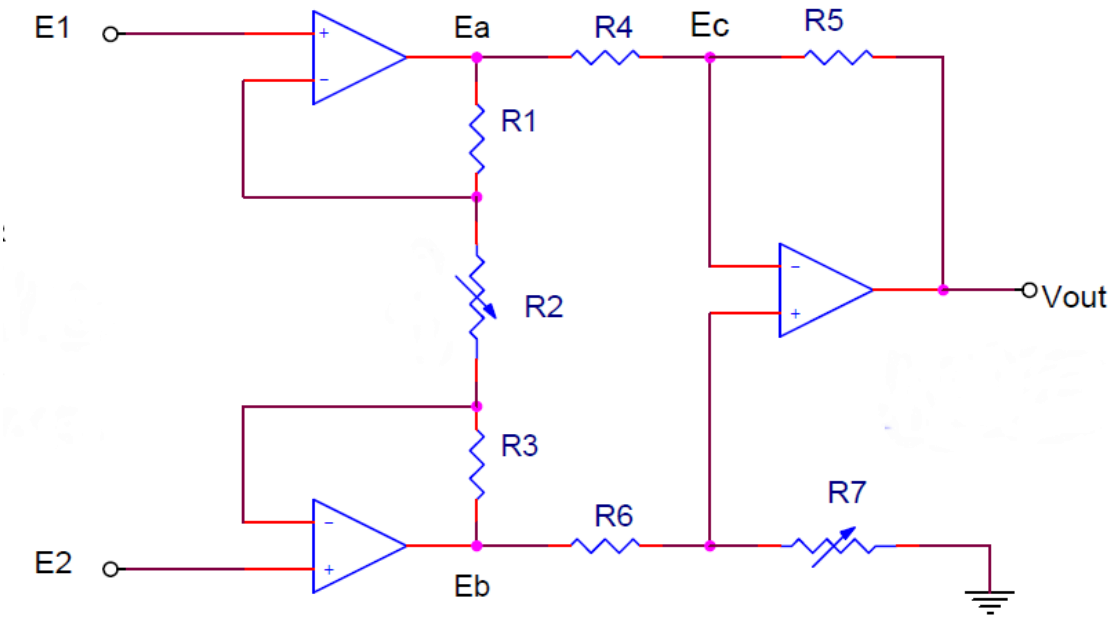


Figura 3. Esquema de un Amplificador de Instrumentación con 3 A.O.

Este circuito se puede ser modelado con las siguientes ecuaciones:

$$\text{Vout} = \text{Ad}\,(\text{E1} - \text{E2}) \qquad \textbf{(3.4)}$$

$$\text{Ad} = -\text{K}\,(1 + \text{G}) \qquad \textbf{(3.5)}$$

$$\text{Siendo K} = \frac{R_5}{R_4} = \frac{R_7}{R_6} \qquad \textbf{(3.6)}$$

$$\text{Y G} = 2\,\frac{R_1}{R_2} = 2\,\frac{R_3}{R_2} \qquad \textbf{(3.7)}$$

3.1.5 Android

Nos encontramos ante un Sistema Operativo basado en un kernel de Linux que fue diseñado en sus inicios para smartphones, pero que ya se ha extendido a muchos otros dispositivos. Entre sus características más notables se encuentran:

- Desarrollo libre y código abierto.
- Se puede usar y "customizar" sin pagar ni pedir permisos.
- Adaptable a cualquier tipo de hardware, como pueden ser: cámaras, electrodomésticos, televisores, etc.
- Las aplicaciones (app) al estar desarrolladas en Java, nos podemos asegurar que podrán ser ejecutadas en cualquier CPU, actual o futura.
- La interfaz de usuario (layout) se basa en el lenguaje XML, lo que permite que se pueda visualizar igual en pantallas reducidas o de mayor resolución.
- Está optimizado para trabajar en sistemas de baja potencia y con poca memoria.

Las aplicaciones que se carguen e instalen en el sistema operativo, son ejecutadas en una máquina virtual Dalvik, la cual está optimizada para este sistema operativo, y traducirá del lenguaje Java en el que se programa la aplicación a código nativo de Android.

Para la programación de la aplicación a desarrollar se usará el IDE Eclipse junto con el SDK (Software Development Kit) de Android 4.2.2, el cual era el más actual cuando se comenzó a trabajar en el proyecto.

3.1.6 Arduino

Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se compone de un IDE, un lenguaje de programación propio y un microcontrolador montado sobre una placa con diferentes entradas y salidas analógicas y digitales. Es una opción interesante para todo aquel que quiera empezar en la electrónica o bien quiera hacer proyectos o diseños de bajo coste, en nuestro caso aprovecharemos esta última cualidad.

Su lenguaje de programación se basa en Wiring, que está basado en C++ y es de código abierto, en cambio su IDE (Integrated Development Environment) se basa en Processing, que tiene sus raíces en Java. Trabaja con microcontroladores de la familia AVR, la mayoría de bajo coste, pero también se han incluido últimamente algunos de mayor potencia.

El código se escribe en lo denominado como "sketch", que consta de dos partes principales: setup() y loop(). El setup() es ejecutado una vez al iniciarse el sketch, y el loop() de forma de continua, siendo un bucle infinito. Todas las declaraciones y definiciones deben hacerse antes del setup().

En el presente proyecto se trabajará con dos placas distintas de Arduino: UNO y Nano. La placa UNO es de uso general, usada en su mayoría cuando se comienza a aprender esta tecnología o para pruebas sencillas. En contrapartida la placa Nano es más limitada en funciones pero tiene un consumo mucho menor, siendo idónea para aplicaciones como la que se quiere desarrollar: un sensor que tenga que ser energéticamente independiente.

1.1. Arduino UNO

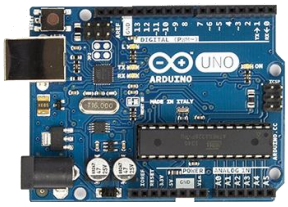


Figura 4. Placa Arduino UNO

Está basada en el microcontrolador ATMega328. Cuenta con 14 pintes digitales de entrada/salida (6 de los cuales pueden ser usados como salidas PWM, "Pulse Width Modulation") y 6 pines analógicos. Incluye también un cristal cerámico de 16 MHz, una conexión USB, una toma de corriente, una cabecera ICSP y un botón de reset.

Lleva incorporada una Memoria Flash de 32 KB en la que cargar el programa desde el IDE. De estos 32 KB, 0.5 KB son usados por el bootloader, por lo que efectivos quedarían 32256 bytes. Además, añade 2 KB de SRAM y 1 KB de EEPROM que pueden ser usados para escribir en ellas desde sus respectivas librerías. Como conexiones externas cuenta con un puerto serie UART ("Universal Asynchronous Receiver-Transmitter"), un puerto SPI y uno I2C (Inter-Integrated Circuit).

Se diferencia de la mayoría de placas Arduino en que tiene un microcontrolador especialmente dedicado para llevar la conexión USB, concretamente ATMega8U

3.1.7 Arduino NANO

Figura 5. Placa Arduino NANO

Al igual que la anterior, está basada en el microcontrolador ATMega328, no cuenta con una toma de corriente, lo cual le deja dos opciones de alimentación: a través del conector Mini-B USB conectado a un PC, o bien alimentación directa a sus pines 5V y GND.

En el caso de los pines, cuenta con los mismos pines analógicos que la placa UNO, pero con dos pines extra analógicos, los cuales en este caso son solo de entrada, no pueden usarse de salida. Cuenta también con el mismo cristal cerámico de 16 MHz, de la cabecera ICSP y del botón de reset.

La memorias son idénticas a las incluidas en la placa UNO, pero en el caso de la placa NANO el bootloader ocupará 2 KB, dejando libres para programar exclusivamente 30720 bytes.

3.1.8 Red de Sensores Inalámbricos (WSN)

Una red de sensores inalámbricos consiste en diversos dispositivos autónomos e independientes, separados entre sí, que tienen como objetivo monitorizar parámetros físicos de su entorno. Comenzaron siendo de uso estrictamente militar, pero ahora se han extendido a múltiples ámbitos, uno muy significativo es el doméstico.

Una de sus características más destacables es su bajo coste y capacidad para funcionar y transmitir la información en tiempo real, lo cual nos impone una condición muy importante en su diseño: su autonomía debe ser lo máxima posible, ya que en la mayoría de casos trabajarán con baterías, en el caso de que no se disponga de acceso a red eléctrica o la posibilidad de usar paneles solares.

En el caso específico de este proyecto, la red será de un sensor con un nodo central que recabará la información, pero este diseño podría ser fácilmente adaptable para usarse con diversos sensores que midieran datos de varios lugares de un domicilio y poder usarlos en una aplicación de Android para informar al usuario de ciertos parámetros interesantes. Un ejemplo de esto podría ser la temperatura de la casa, sacando la media de varios sensores de temperatura repartidos por la casa, y tener un microcontrolador sencillo y de bajo consumo conectado al aire acondicionado o calefacción para poder desde nuestro terminal móvil regular la temperatura de la casa cuando no se esté en ella.

3.1.9 Estándar IEE 802.15.4

Este estándar define el protocolo de las capas OSI superiores y define los perfiles de aplicación que pueden compartirse entre diferentes fabricantes. Las dos primeras capas las define el estándar y las restantes ZigBee.

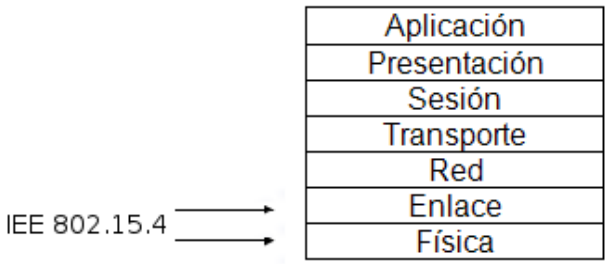


Figura 6. Esquema de Capas OSI

3.1.10 ZigBee

Tecnología de comunicación inalámbrica que opera en la banda de 2.4 GHz, siguiendo el estándar IEEE 802.15.4. En Europa permite también operar a 868 MHz. Su velocidad máxima de transmisión es de 250 kbits por segundo.

Su mayor potencial es el consumo reducido de sus dispositivos, siendo estos óptimos para tareas como la que nos atañe, debido a que nos interesa reducir todo lo que podamos el consumo de nuestro sensor y la placa de Arduino a la que irá conectada.

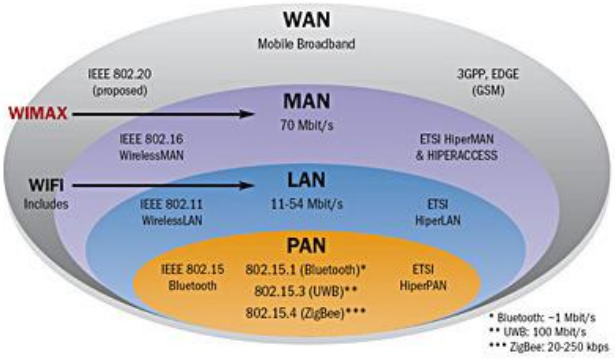


Figura 7. Diferentes tecnologías que trabajan en 2.4 GHz

A pesar de que una red ZigBee se puede componer de hasta 254 nodos, para el desarrollo del presente trabajo, tan solo se usarán 2 nodos, pero se deja abierta la posibilidad de ampliar la red colocando sensores destinados a otras tareas por el domicilio.

3.1.11 WiFi

Protocolo de comunicación inalámbrica, el más extendido en el mundo, y gracias al cual podemos acceder a Internet sin necesidad de cables desde nuestros portátiles, tablets o smartphones, pudiendo añadir a estos dispositivos muchas más posibilidades de comunicación y de formas de interactuar con las personas

Trabaja en la banda de 2.4 GHz, y con el último estándar, el IEEE 802.11n llega a unas velocidades de 300 Mbit/s. Permite dar acceso a multitud de dispositivos y tener fácil acceso de ellos a través de las direcciones IP, identificaciones de cada dispositivo.

3.1.12 HTTP

Hypertext Transfer Protocol, protocolo usado para la mayoría de transacciones de datos en Internet. Define la sintaxis y semántica que usan los clientes y servidores de la web para comunicarse, siguiendo el esquema de petición-respuesta. La información que se transmiten de uno a otro se la identifica mediante una URL (Uniform Resource Locator).

HTTP no guarda datos de sesión de anteriores conexiones, es por ello necesario utilizar las conocidas como "cookies" si se quiere tener información de anteriores acciones entre cliente-servidor.

La forma de obtener un recurso es mediante una petición GET, la cual tiene la siguiente forma:

"GET /<recurso> HTTP/1.1 Host: <nombre_dominio> [Línea en blanco]"

Una petición GET también puede ser empleada para enviar parámetros:

"GET /<recurso> ?page=main&lang=es HTTP/1.1 Host: <nombre_dominio> [Línea en blanco]"

Esta última petición añade los parámetros de "main" para el campo "page" y de "es" para el campo "lang".

3.1.13 SPI

SPI son las siglas en inglés de Serial Peripheral Interface, un estándar de comunicaciones muy utilizado para transmitir información entre circuitos integrados en equipos electrónicos. Se trata de una comunicación serie, por lo que envía bits uno a uno, siendo además del tipo Maestro-Esclavo, teniendo el Maestro el control del Reloj.

Incluye una línea de Reloj (SCLK), dato de entrada (MISO), dato de salida (MOSI) y un chip select (SS), éste último pin es activo por nivel bajo. El pin SS es lo que más caracteriza este estándar, permitiendo habilitar según se requiera diferentes periféricos (sólo se puede tener uno habilitado en cada instante) y asir tener una comunicación entre un maestro y varios esclavos.

Su mayor ventaja es los pocos pines necesarios para su funcionamiento, y la posibilidad de extenderse a múltiples esclavos con suma facilidad, ya que sólo es necesario añadir una línea SS extra por cada esclavo.

3.1.14 LM35

Sensor de temperatura con una sensibilidad de 10mV/ºC, con un rango de medida que abarca desde los -55ºC hasta los 150ºC.

Su mayor ventaja es que la salida obtenida es lineal a la temperatura, con una precisión en sus mejores versiones de hasta +- 0.2 ºC, pudiendo además operar con tensión bipolar o unipolar, con un margen de 4 a 30 voltios. Aparte, cuenta con una baja impedancia de salida y no necesita de ningún tipo de circuito adicional para ser calibrado. Por último, destacar su bajo precio.

Además de las anteriores características, su consumo es acorde a lo que necesita para este proyecto, pues ronda los 60 uA, lo cual es una doble ventaja, pues el efecto de autocalentamiento es mínimo.

3.2 GESTIÓN DEL PROYECTO

El planteamiento del proyecto queda explicado con lo expuesto anteriormente, pero para poder llevarlo a cabo se requiere una buena distribución, estudio y desarrollo de las diferentes partes que lo componen, prestando especial atención a la comunicación entre las diferentes tecnologías que se decidan utilizar.

A continuación se hará una introducción y desarrollo de cada parte en las que se ha dividido el proyecto y cómo fueron planteadas en un principio. En futuros apartados se verá el resultado final, y se hará una comparativa de la evolución sufrida.

3.2.1 LEGISLACIÓN

El primer tema que se trató, antes de comenzar a mirar componentes electrónicos, programación o cualquier otra cosa fue el tema de la regulación de la instalación de recipientes de G.L.P. según la legislación de España, debido a que se plantea trabajar con componentes electrónicos sobre algo potencialmente explosivo, la precaución es lo primero.

En España la norma vigente sobre instalaciones de Gases Licuados de Petróleo (G.L.P.) viene regida por el Real Decreto 919/2006, aunque también se han tomado en cuenta las consideraciones propuestas y los datos de la NTP 209. Ambos documentos están referidos en los Anexos.

De la norma NTP 209 se han extraído los siguientes datos:

- Densidad del butano licuado envasado.
- Nombre del envase y material de fabricación.
- Presión del G.L.P. en el interior del envase.
- Recomendaciones sobre electricidad y electrónica en su lugar de almacenamiento: *Las botellas distarán como mínimo 0,30 m. de los interruptores y de los conductores eléctricos, y de 0,50 m. de los enchufes eléctricos (NTP 209 – pag.11).*
- Peso del envase, cantidad de G.L.P. que contiene, y su tolerancia.

Dado que en el sistema a desarrollar, la electrónica que estará cerca de la bombona será toda interna en la estructura del sensor, no debemos preocuparnos por la recomendación de la NTP 209, debido a que esta misma estructura aislará la bombona de cualquier posible incidencia.

Añadir, además, que la placa Arduino del sensor trabajará en términos de baja potencia, reduciendo así el riesgo de incidentes.

SENSOR DE PESO

La masa de la bombona está regulada por la legislación, aunque actualmente existen dos tipos de recipientes: la bombona naranja que se ha usado durante muchos años, y unas nuevas de acero inoxidable, mucho más ligeras.

La primera tarea fue comprobar en la normativa el peso de cada tipo de bombona, y sus tolerancias.

El modelo UD - 125 (bombona naranja) tiene una masa de 26.4 kg llena, siendo 12.5 kg de gas butano, por lo que vacía tendría 13.9 kg. La de acero inoxidable, por el contrario, es de tan solo 18 kg llena, cuenta también con 12.5 kg de gas butano, lo cual deja 5.5 kg vacía. También se indica que la tolerancia es de $\pm$ 0.9 kg, aunque para este proyecto no lo tendremos en cuenta, debido a que nuestras medidas serían informativas, no determinantes para tomar una decisión crítica.

Una vez se conoce qué se ha de medir, ha de plantearse cómo se ha de llevar a cabo la medida: se usarán Galgas Extensiométricas metálicas para el desarrollo del sensor, debido a que sus características se presentan como idóneas para el trabajo pensado y por la sencillez de su implementación.

Como primera idea se pensó en usar una base o soporte para la bombona, con tres patas y pegadas en esas patas las galgas. Esto presenta varias ventajas, como es que es posible compensar fácilmente la desviación que puede haber al no colocar la bombona en el centro del soporte obteniendo la media de las tres medidas, y colocando dos galgas por pata se puede lograr una compensación en temperatura y aumentar la sensibilidad del sensor. Se escogió el aluminio como material para las patas.

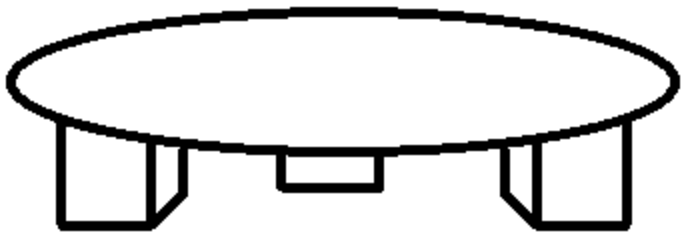


Figura 8. Esquema de la idea inicial para el sensor de peso

Igualando las fórmulas (3.1) y (3.2) se llega a: $\frac{F}{S} = E\,\varepsilon$

E = 70 GPa (aluminio), ε máximo de las galgas escogidas es 10000 με. Se escogerá la fuerza máxima que registrará el sistema: la bombona llena, 26,4 kg.

Con estos valores se puede conocer la superficie que se requerirá de las patas $S = \frac{1}{3}\frac{F}{E*\varepsilon}$, se divide por 3 debido a que se desea conocer la superficie de una pata, no del conjunto. Al sustituir los datos resulta: S = 0.23 mm^2.

Se puede apreciar que se necesitarían unos soportes ridículamente pequeños, por lo que este planteamiento fue descartado, por ser inviable.

La siguiente opción a considerar fue usar una lámina de flexión, poniendo el centro de la bombona en el centro de la misma y así repartir la fuerza del peso a partes iguales. La idea se extrajo del sistema que usan las básculas comunes domésticas, las cuales hacen uso de una estructura metálica que distribuye la fuerza que se ejerce en toda la superficie en un único punto, siendo este punto la lámina de flexión. Para ilustrar esto se ha incluido una imagen (Figura 27) donde se puede apreciar dicha estructura remarcada en rojo.

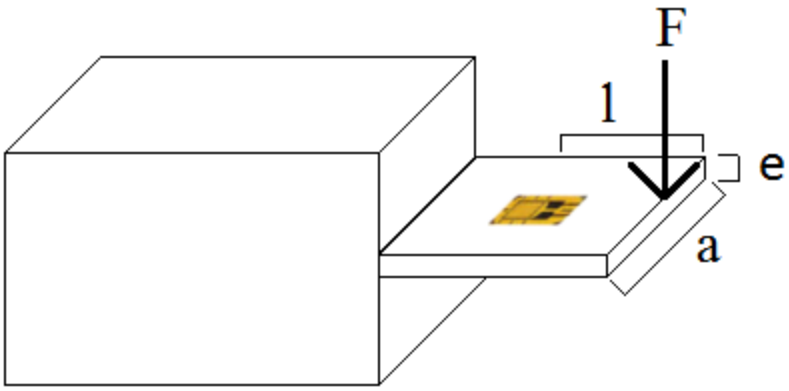


Figura 9. Esquema de una lámina de flexión.

La deformación que sufre la galga en una lámina de flexión puede ser calculada atendiendo a la siguiente fórmula:

$$\varepsilon l = \frac{6\,F\,l}{E\,a\,e^2} \qquad \textbf{(3.8)}$$

Siendo a, e, F y l los mostrados en la figura anterior, εl la deformación de la galga y E el módulo de Young del material.

Para la lámina se usará aluminio, siendo su módulo de Young, conocido anteriormente, de 70 GPa. Sabiendo esto, se ha diseñado la lámina para tener un espesor "e" de 2 mm, una anchura "a" de 5 cm, y se usará una galga extensiométrica como la expuesta anteriormente, capaz de soportar hasta 10000 με. Sustituyendo los datos anteriores, obtenemos que la galga debe ser colocada a una distancia longitudinal en la lámina de 9 cm.

Esta vez los resultados sí que se encuentran dentro de lo razonable, por lo que finalmente se optó por esta como estructura definitiva para el sensor.

3.2.2 SENSOR DE TEMPERATURA

Durante el desarrollo del proyecto se planteó la posibilidad de medir el valor de temperatura alrededor de la bombona de butano, siendo este dato interesante de conocer debido a que la variación de la temperatura exterior, y por tanto de la interior de la bombona, condiciona el porcentaje mínimo de gas que se necesita para su funcionamiento, siendo este más bajo para temperaturas elevadas y más alto para temperaturas bajas.

Este razonamiento puede ser demostrado mediante la ecuación de estado de los gases:

$$P\,V = n\,R\,T \qquad \textbf{(3.9)}$$

Dado que el volumen (V) y la constante universal de gases (R) son constantes, quedan tres variables: presión (P), número de moles del gas (n) y la temperatura (T).

Al reducirse el número de moles del gas en el envase (al usarse) la presión en el recipiente disminuye, hasta llegar a un punto crítico en el cual el calentador no puede prender. Este punto crítico puede alcanzarse también si la temperatura disminuye, para un número de moles constante, la presión sería menor.

Para poder efectuar estos cálculos correctamente y desarrollar un estudio teórico sobre este fenómeno, haría falta efectuar pruebas empíricas, debido a las diferencias que existen entre los diferentes calentadores en el mercado, por lo cual se ha optado por dejar indicada la medida de temperatura para que el estudio pueda llevarse a cabo como ampliación futura del proyecto.

Como sensor de temperatura se ha optado por el circuito integrado LM35A, el cual proporciona una salida directamente proporcional a la temperatura ambiente, simplemente con una alimentación unipolar y que se adapta a las especificaciones de nuestro sistema.

3.2.3 ARDUINO: MEDIR Y SLEEP MODE

Una vez las medidas han sido obtenidas con el circuito diseñado, serán introducidas a la placa Arduino NANO por los pines analógicos 1 y 2, para masa y temperatura respectivamente.

La lectura de estos puertos es inmediata con la función analogRead(pin analógico), guardando cada dato en una variable integer diferente, para su posterior conversión al valor físico deseado y transmisión a la otra placa Arduino.

Antes de investigar y desarrollar cómo realizar esa transmisión de datos, cabe destacar la importancia de que la placa Arduino pueda entrar en modo sleep, donde se reduce drásticamente su consumo, elevando sustancialmente la autonomía del sistema.

3.2.3.1 CONTROL DEL MODO SLEEP

Según las librerías incluidas en Arduino ("sleep.h" y "power.h"), llevar a la placa al modo sleep es sencillo, pero se debe tener muy en cuenta la forma en que será despertada, pues cada forma de despertar nos permite sumirlo en un modo sleep más profundo, el cual nos permite ahorrar en consumo energético y aumentar la autonomía, más adelante se profundizará en los diferentes métodos.

El procedimiento a seguir siempre cuando se quiera incluir el acceso al modo sleep en el programa, deberá ser el siguiente:

set_sleep_mode(mode) – Configura el ATMega168 para el modo sleep especificado.

sleep_enable() – Habilita el modo sleep para que pueda entrarse en él.

Sleep_mode() – Entra en el modo sleep. Antes de que llamemos a esta función, el sistema para despertar al microcontrolador se debería de haber puesto en marcha.

sleep_disable() – Deshabilita el modo sleep, para que no se vuelva a poder entrar en él.

A continuación se detallarán los diferentes métodos para despertar al microcontrolador:

- Interrupción Externa: Se seleccionará el modo "PWR_DOWN", se dejará un pin configurado como INPUT a la escucha de que la señal quede en LOW o HIGH, según sea configurado con la función detachInterrupt() la cual deberá ser accedida dentro de la función que estará a la escucha: pinNInterrupt(void), siendo N el número de pin que se desee usar para la escucha.

  Las ventajas de emplear este método son muchas: no consumimos casi energía entrando en el modo "PWR_DOWN", y se puede prolongar el modo sleep mucho más que lo que permiten los otros métodos. Como contrapartida cabe destacar que este método requiere de un circuito externo de control, normalmente con condensadores que se carguen/descarguen.
- Temporizador Interno: La placa Arduino NANO cuenta con 3 Temporizadores diferentes: 0, 1 y 2. El Temporizador 0 y el 2 tienen una longitud de 8 bits, llegando a alcanzar un tiempo máximo de 16.4 ms, mientras que el Temporizador 1, al ser de 16 bits, nos permite alcanzar los 4.1 s. Estos cálculos han sido llevados a cabo suponiendo el reloj de 16 MHz y con un reescalado máximo, siendo este 1:1024, siguiendo la fórmula siguiente:

  TimeoutPeriod = (1/16MHz) * Prescaler * 2^(timer bit count)

- Temporizador Watchdog: Es un temporizador propio del microcontrolador, el cual funciona con un reloj independiente de 128 kHz. Permite entrar en el modo sleep más profundo: "PWR_DOWN", pudiendo llegar a dejar durmiendo el sistema hasta 8 s, lo cual es casi el doble de lo obtenido como máximo por los Temporizadores internos. Para introducirlo en nuestro programa se deberá de incluir su propia librería: "wdt.h". Se implementará como un Temporizador normal en el código.

Se puede apreciar claramente, que la opción de la interrupción externa es la que mejor se puede adaptar las necesidades del sistema, pero al no disponer del material necesario para llevarlo a cabo, debido a que se estuvo barajando la posibilidad de incluir el integrado IC555 (circuito integrado que actúa como temporizador regulable, lo cual es idóneo para el proyecto que se está desarrollando. Pero al no contar con este material, se decidió utilizar la opción del Temporizador Watchdog, siendo la segunda mejor opción.

3.2.4 COMUNICACIÓN INALÁMBRICA ENTRE LAS DOS PLACAS ARDUINO

El haber solventado el problema de la autonomía del sistema, lo cual era uno de los principales objetivos a cumplir, permite que la atención pase a la forma de comunicarnos.

La idea original del trabajo fue emplear la tecnología ZigBee para realizar esta comunicación, pudiendo aprovechar las ventajas que ofrece para este tipo de aplicaciones de bajo consumo y que es fácilmente implementable en Arduino, debido a la gran cantidad de módulos y de librerías que existen.

En la sección de "Elección de Material" se detallará en profundidad cómo se llevó a cabo la elección de los módulos de comunicación inalámbrica, y en la sección de "Desarrollo y Resultados" cómo fueron estos implementados en los programas de cada placa de Arduino para su correcto funcionamiento.

3.2.5 COMUNICACIÓN DE LA PLACA ARDUINO CON EL SMARTPHONE

Este fue uno de los primeros asuntos que se plantearon y que se debía resolver para continuar avanzando en el desarrollo del proyecto. Se estudiaron dos posibles tecnologías a uasr:

- WiFi
- Bluetooth

A continuación se expondrán las cualidades de cada una y por cual se optó como definitiva.

3.2.5.1 WIFI

Proporciona una gran velocidad de transmisión de datos, una buena cobertura en un domicilio estándar y la posibilidad de conectar varios equipos con facilidad.

Trabaja en la banda de 2.4 GHz, y con el último estándar, el IEEE 802.11n alcanza una velocidad de 300 Mbit/s. El principal problema viene por su consumo, debido a que es bastante elevado por la gran cantidad de datos que permite transmitir.

3.2.5.2 BLUETOOTH

Es una tecnología que trabaja también a 2.4 GHz, en su versión 4.0 alcanza los 24 Mbit/s, y según el modelo se obtienen alcances de centenares o decenas de metros, con un consumo menor que en sus anteriores versiones, pudiendo competir con el WiFi en este sentido.

3.2.5.3 ¿POR CUÁL SE HA OPTADO?

En vista a las cualidades de cada uno, se decidió optar por comunicar la placa Arduino con el smartphone mediante WiFi, pero no directamente, si no usando un servidor donde almacenar los datos por parte de la placa Arduino y desde allí recogerlos con la aplicación de Android.

Esto está pensado no solo para el proyecto que se plantea, si no para poder aplicarlo a cualquier sistema de medida, pudiendo ampliarse con el uso de una base de datos en el servidor y tener diferentes usuarios, cada uno con sus medidas y sus datos, y que puedan acceder a ellos con independencia de su situación geográfica.

El Bluetooth limitaba la consulta a cuando el usuario estuviera en su domicilio, a parte de que no es común tener el Bluetooth activado en el smartphone, y sería obligar al usuario a activarlo para consultar el nivel de gas que le queda.

Esta solución plantea otra etapa en el diseño del trabajo, debe usarse un servidor para guardar los datos, lo cual será la siguiente tarea que se explicará.

3.2.6 SERVIDOR WEB

Las especificaciones que se requerían del servidor no eran para nada elevadas en términos de capacidad de almacenamiento ni capacidad de computación, debido a que la tarea y los datos con los que se trabajan son de tamaño considerablemente pequeño.

Para ello se tomó la decisión de adquirir un hosting gratuito donde subir los ficheros con los que llevar a cabo la tarea para la cual se pensó en utilizar el servidor: subir y descargar los datos de las medidas. La URL del dominio asignado es la siguiente: http://proyectoteleco.esy.es/ .

Los archivos que se generarán y programarán para poder realizar la tarea podrán ser subidos al servidor mediante el protocolo FTP, y las peticiones para la subida y bajada de la información medida se realizarán a través de HTTP GET, teniendo campos diferentes para cada medida.

Se pensó también en añadir una interfaz simple para poder consultar directamente en la web del servidor los valores de la última medida que se subió al servidor.

3.2.7 APLICACIÓN ANDROID

Dado que al comienzo del desarrollo del proyecto no se tenían conocimientos de programación para este lenguaje, la primera tarea fue estudiarlo, mediante tutoriales y cursos en Internet.

Durante los primeros días de estudio se pudo describir mejor las especificaciones que se querían de la aplicación:

- Debía tener una interfaz sencilla e intuitiva, botones grandes y claros, texto corto pero explicativo.
- Las consultas al servidor no debían de ser constantes para no agotar la batería, debían de ser reguladas.
- Debía de buscarse una forma de representar gráficamente el nivel de gas restante en el recipiente.

A partir de estas simples ideas se desarrolló cómo debería ser la aplicación, dejando abierta la opción de que fuese cambiando con el avance del conocimiento de programación en Android.

3.3 DIAGRAMA TEMPORAL

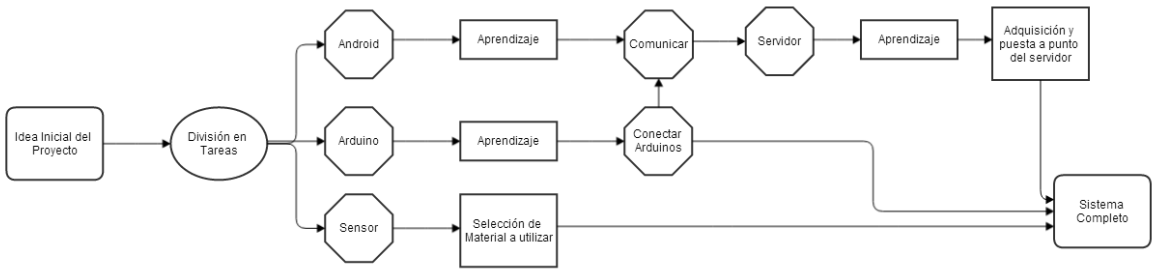


Figura 10. Diagrama Temporal del proyecto

Capítulo 4. DESARROLLO Y RESULTADOS

En este capítulo se tratará en profundidad el desarrollo de cada una de las partes que componen el proyecto, los cambios sufridos y el resultado final del proyecto en conjunto.

4.1.1 SENSOR DE PESO

Una vez se ha decidido la estructura que seguirá el sensor de peso es el momento de dedicar tiempo a hacer el análisis teórico del circuito que será usado para medir el peso de la bombona de butano.

La primera tarea es el diseño del puente de Wheatstone que se usará para el sensor. Se ha decidido utilizar dos galgas: una colocada en la parte superior de la lámina y otra en la inferior, con el fin de poder realizar la compensación en temperatura en el puente de Wheatstone, además de aumentar la sensibilidad del sistema.

Se usarán dos resistencias (R_1 y R_2) con un valor idéntico, para que V_2 sea Vcc/2, mientras que R_{G1} y R_{G2} representan las dos galgas montadas. La salida del puente Vo = $V_1 - V_2$ puede ser escrita como

$$\mathrm{Vo} = \frac{\mathrm{Vcc}}{2}\left(\frac{R_0(1-x)}{R_0(1-x)+R_0(1+x)} - 1\right) = \frac{Vcc*x}{2} \qquad \textbf{(3.10)}$$

Si se sustituye $x = k\ \varepsilon_l$, F = 9.8 M (kg) y atendiendo a la fórmula (3.8), se obtiene que:

Vo = 1,89 10^{-3} M (kg). Con este dato puede hacerse una idea del nivel de tensión con el que trabajaremos, milivoltios, por lo que no será necesario amplificar la señal, ya que la sensibildad mínima de la placa Arduino NANO se puede conocer de la siguiente manera: para el ADC de la placa la tensión de referencia se puede cambiar como una entrada externa, por lo que se debería a justar a su valor más bajo: 1,2 V, extraído del catálogo. La conversión del ADC se hará con 10 bits por lo que el dato obtenido se encontrará entre 0 y 1023, si se saca la relación se ve que es aproximadamente de 1 mV la mínima tensión que puede percibir el Arduino. Se ha elegido utilizar un Amplificador de Instrumentación con ganancia de 10 para el acondicionamiento de la señal, dejando la salida y la sensibilidad de la siguiente manera:

Vout = M (kg) 18,9 mV, lo cual lleva a tener una sensibilidad de:

S = 18,9 mV/kg.

El diseño del Amplificador de Instrumentación se llevará a cabo siguiendo las ecuaciones (3.4) (3.5) (3.6) (3.7). Escogiendo K = 1 y G = 9. Deja la elección de resistencias de la siguiente manera:

R5 = R7 = 1 kΩ, R4 = R6 = 1 kΩ, R1 = R3 = 1 kΩ, R2 = 111.11 Ω (la cual se conseguirá con un potenciómetro de 1 kΩ).

El esquema final del diseño del sensor y su acondicionador se puede observar en su correspondiente apartado: (Figura 25).

Para la interpretación de la medida de la medida por parte de la placa Arduino, se deberá realizar el siguiente cambio en el programa (con una tensión de referencia de 1.2 V):

Masa = Valor medido 1200/1023 189/1000

4.1.2 SENSOR DE TEMPERATURA

El montaje del sensor de temperatura será tan sencillo como conectar el circuito integrado LM35A a la alimentación que proporciona el Arduino NANO: Vs = +5V y GND. La salida irá conectada al pin analógico 2 de la placa Arduino para ser medida. Dado que la salida del LM35A sigue la siguiente expresión: Vout = 10 mV/ºC, y que como se ha comentado antes, usaremos una tensión de referencia de 1.2 V en la placa, se seguirá un procesión de transformación de la tensión medida para obtener la temperatura similar al anterior:

Temperatura = Valor medido 10/1000 1200/1023 (ºC)

Se ha comentado que la tensión de referencia que será usada en el Arduino NANO es de 1.2 V, pero para obtener esa tensión se deberá montar un simple divisor de tensión tal cual aparece en (Figura 26), conectando su salida a la entrada AREF de la placa.

Con este simple cambio en la tensión de referencia de la placa Arduino se consigue la medida de la temperatura deseada. Además, según el datasheet, alimentando a +5V el circuito integrado LM35 puede trabajar en zona lineal de +2 ºC a 150 ºC, margen más que suficiente para la mayoría de los casos que se pueden plantear, dejando fuera solo a las zonas más frías.

Una vez tenemos las dos medidas acondicionadas y recibidas por la placa Arduino NANO, se dan por finalizada la parte de los sensores y se pasa a continuación a la parte de comunicación.

4.1.3 COMUNICACIÓN INALÁMBRICA ENTRE LAS DOS PLACAS ARDUINO

Tal y como se planteó en el apartado de Gestión del Proyecto destinada a esta parte del trabajo, se pensó en utilizar módulos que funcionasen con la tecnología ZigBee para realizar esta comunicación.

En el apartado de Elección de Material se detallará las razones por las que se descartó utilizar módulos XBee para la comunicación, y que al final se optó por una solución más barata y que ofrecía características similares: un protocolo pseudoZigBee (basado en el estándar 802.15.4 pero lo suficientemente diferente como para que la diferencia de precios de los módulos fuese considerable).

El módulo escogido fue el nRF24L01, el cual cuenta con múltiples ejemplos de uso y librerías por Internet, lo cual facilita su uso en proyectos.

Estos módulos funcionan a 2.4 GHz, al igual que el estándar ZigBee, utilizando modulación GFSK, a una tasa que puede ser elegida entre 250 kbps, 1 o 2 Mbps, superior a la que se logra con ZigBee.

Su alcance en espacios interiores es de entre 10 y 20 m, lo cual es más que suficiente para las necesidades del proyecto, mientras que su consumo se especifica en la siguiente tabla, para una tasa de 2 Mbps y una potencia de transmisión de 0 dBm.

	Transmisión	Recepción	Standby	Sleep Mode
Corriente (A)	13.5 m	11.3 m	26 µ	900 n

Tabla 1. Datos de consumo del nRF24L01 extraídos del catálogo.

Su comunicación con la placa Arduino se realiza mediante SPI, contando para ello con los pines de CSN (Slave Select), SCK, MISO y MOSI. La comunicación de estos pines y la alimentación para el módulo se detallará más adelante.

La programación de la transmisión y recepción usando este módulo se basará en una librería con nombre "RF24", incluida en los Anexos. Se definirán dos direcciones diferentes para la comunicación: una de transmisión y otra de recepción, y debido a que cada módulo sólo actuará como uno de estos dos, no será necesario variar sus direcciones. El esquema siguiente muestra cómo se organizarán ambos programas:

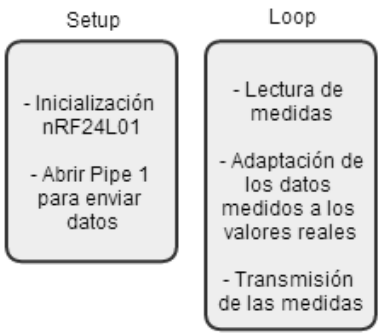


Figura 11. Sketch Arduino NANO

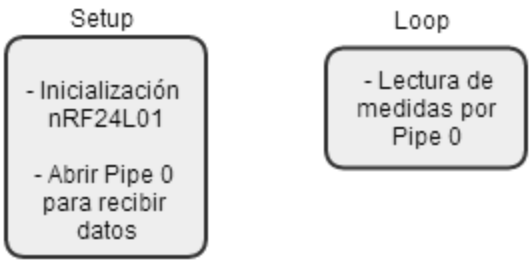


Figura 12. Sketch Arduino UNO

Las pruebas de comunicación entre ambas placas Arduino por medio de los módulos nRF24L01 son satisfactorias, pudiendo cerrar esta tarea y centrarse en la siguiente.

4.1.4 IMPLEMENTACIÓN DEL TEMPORIZADOR WATCHDOG

En la placa Arduino NANO, además del esquema anterior deberán ser añadidos los siguientes bloques, los cuales permiten al sistema entrar al modo sleep profundo cada 8 segundos, aumentando considerablemente su autonomía.

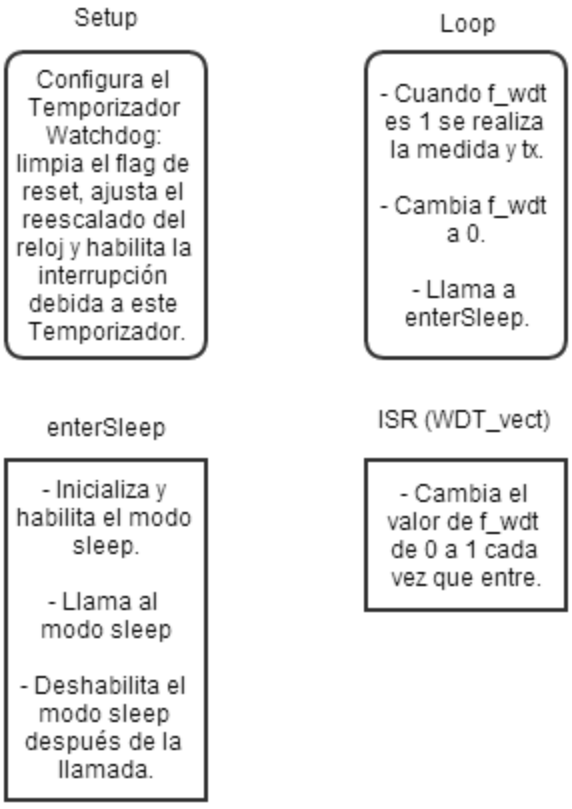


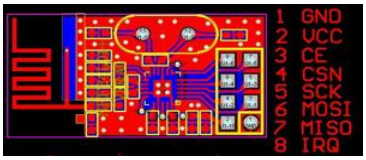
Figura 13. Diagrama Temporizador Watchdog en NANO

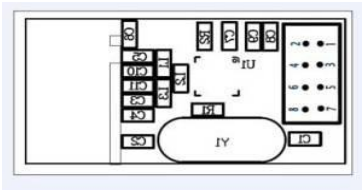
Este diagrama se da separado ya que es fácilmente sustituible por una Interrupción Externa, como fue comentado en su punto correspondiente, y será uno de los temas a tratar en las propuestas de trabajo futuro.

4.1.5 CONEXIÓN NRF24L01 Y SHIELD WIFI CON ARDUINO UNO

Para la placa UNO se tendrán dos periféricos para conectar a la placa Arduino, los cuales se comunican con la placa mediante el mismo protocolo, SPI, el cual soporta la comunicación entre un maestro y varios esclavos, controlándola mediante la señal "Slave Select" (SS o CSN).

El Shield Wifi irá colocado encima de la placa Arduino, sin necesidad de ningún tipo de conexión especial, la placa viene ya diseñada para encajar y funcionar de manera óptima. En el caso del módulo NRF24L01 no será ese el caso, pues es en el catálogo donde se encuentra la distribución de los pines, y con ello conocer cómo conectarlo a la placa Arduino.





Figuras 14. Layout del nRF24L01 ***Figura 15. Pines del nRF24L01***

Aquí se puede apreciar el layout del módulo y las señales que corresponden a cada pin, además de su distribución física.

Por suerte los pines de comunicación SPI de las dos placas Arduino están distribuidos de igual manera, por lo que se detallará la conexión a la placa UNO y de igual manera se haría con la NANO.

Señal	Recepción	Standby
GND	1	GND
VCC	2	3V3
CE	3	9
CSN (SS)	4	10
SCK	5	13
MOSI	6	11
MISO	7	12
IRQ	8	(No conectado)

Tabla 2. Conexiones de los pines entre nRF24L01 y la placa Arduino

Por suerte las librerías de ambos periféricos ofrecen la posibilidad de modificar el pin que usará como Slave Select, permitiendo asegurar que no haya interferencias y facilitando la tarea de conexión.

Se ha optado por modificar el SS del módulo RF en su conexión con la placa Arduino UNO, cambiándolo del pin 10 por defecto al pin 8, esto se realiza en la llamada de configuración que se hace a la clase de la librería, como se puede apreciar en el código incluido en el Anexo.

4.1.6 SERVIDOR WEB

En el dominio del proyecto se encuentra un directorio con el nombre de "medidas", en la cual están los archivos con los que se ha trabajado para dar el servicio de subir y leer los datos. La lista de archivos es la siguiente:

- datos.html
- jquery-2.0.3.min.js
- mostrar_masa.php
- mostrar_temp.php

- masa_datos.txt
- temp_datos.txt
- sensor.php
- style.css

masa_datos.txt y temp_datos.txt son los ficheros donde guardaremos las medidas subidas por la placa Arduino. Y serán también los archivos a los que acceda la aplicación en Android para leer las medidas y mostrarlas por pantalla.

Los demás archivos los usaremos para mostrar las medidas en una página Web, lo cual vendrá a través de datos.html, que recurrirá al código de mostrar_masa.php y mostrar_temp.php para leer los archivos de texto. A su vez, he añadido style.css para mejorar el estilo visual del texto mostrado, el resultado puede apreciarse en la siguiente captura de lo mostrado en un navegador.

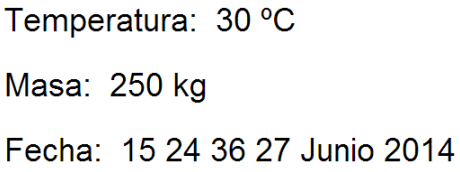


Figura 16. Captura de navegador

Por último, tenemos el archivo jquery-2.0.3.min.js, el cual nos dará el soporte para las funcionalidades JQuery de JavaScript para poder refrescar la información en la página html cada segundo.

4.1.7 CONFIGURACIÓN WIFI Y SUBIDA DE DATOS AL SERVIDOR

El Shield WiFi que se adquirió para el desarrollo de este trabajo no contaba con un catálogo propio proporcionado por el fabricante, la única solución que se encontró para solventar este problema fue realizar una búsqueda exhaustiva por Internet con el fin de encontrar alguna referencia a un catálogo o librería desarrollada para dicho Shield.

Hubo buena fortuna con la búsqueda, pues la librería "Adafruit_CC3000" se aseguraba que era totalmente funcional para el Shield adquirido, por lo cual fue esta la que se usó en el desarrollo del programa para la placa Arduino UNO.

Los primeros pasos para la configuración y correcto funcionamiento del Shield WiFi fueron encaminados a comprobar si verdaderamente era funcional con dicha librería: se ejecutó un test que se incluía como ejemplo y los resultados fueron satisfactorios, todo funcionaba perfectamente.

Con el lastre de encontrar librería eliminado, tan sola resta la programación que interesa para el trabajo: conectarse a una red WiFi doméstica y ser capaz de subir los datos de medidas a un servidor web.

El programa para la placa Arduino UNO seguirá el siguiente esquema:

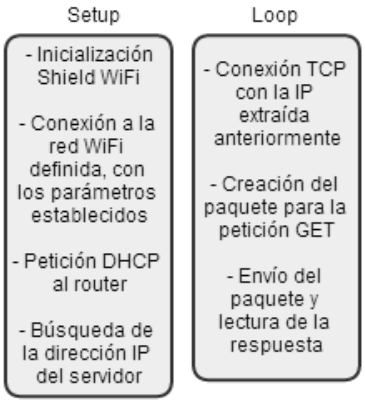


Figura 17. Sketch Arduino UNO

Una vez cargado se comprueba su correcto funcionamiento y cómo el servidor recibe los datos correctamente. Con esta parte funcionando tan solo resta la aplicación en Android.

4.1.8 APLICACIÓN ANDROID

La aplicación a desarrollar contará con un solo layout, donde se podrán apreciar tres botones: dos para que el usuario pueda elegir el tipo de bombona que use en su hogar, y otro para actualizar la información desde el servidor. En pantalla se mostrará el tipo de bombona que se haya seleccionado y el nivel del gas actual junto a la temperatura ambiente, o bien, la última medida que se tomó.

En el caso de que no se haya seleccionado tipo de botella se mostrará un mensaje indicándolo, siendo esto imprescindible para que se llegue a mostrar el nivel de gas o temperatura medida.

Aparte del mensaje para el nivel de gas restante, o bien que no se haya seleccionado bombona todavía, en el layout se encuentra otro TextView que indicará la temperatura de la última medida.

Se ha elegido como forma de representar el nivel de gas restante una barra progresiva, teniendo su mínimo en 0 y su máximo en 100, siendo por tanto el valor en tanto por cien lo que el usuario verá en pantalla. Otra opción podría ser hacer uso de un Animation Drawable, y partiendo de una imagen de una bombona vacía mostrar el porcentaje restante en forma de líquido en su interior.

La petición al servidor será del tipo Http Get, siendo el servidor un hosting gratuito que se ha dispuesto para la presentación del proyecto donde se han subido diversos archivos que permitirán visualizar y descargar los valores de las medidas.

Se harán dos peticiones, una para la temperatura y otra para la masa. Cada una accederá a una URL diferente, donde estará un archivo de texto con su respectivo valor de medida.

Dado el sistema de servicios de Android, la petición Http Get deberá hacerse en un hilo diferente del principal de la aplicación. Se ha optado por usar una AsyncTask, para liberar al sistema de recursos mientras se procesa la petición y la información recibida es tratada.

Una vez hecha la petición, habrá que tratar la String recibida y sacar de ahí los valores que nos interesan, pero dado que al servidor sólo subimos el valor de medida, basta con convertir la String del mensaje recibido a un integer, y este adecuarlo para representarlo en tanto por cien en la ProgressBar del layout.

Para hacer este cambio para su representación, se usará un sistema lineal basado en una recta, debiendo de hacer uno para cada tipo de bombona.

4.1.9 MEJORA: USO DE LA FECHA DE LA MEDIDA

Una mejora que se pensó en incluir en el sistema fue la adquisición por parte de la placa Arduino UNO de la fecha en el momento de subir los datos de medida al Servidor, para incluirla como

datos y así que nuestra App pueda mostrar al usuario hace cuánto que no se realiza una medida, para que pueda identificar si el sistema se ha quedado sin pilas en la base del sensor, o bien cualquier fallo en el servidor o en la placa Arduino.

Para lograr esto se pensó en hacer una petición al Servidor NTP (Network Time Protocol), el cual nos permite obtener el tiempo transcurrido desde el 1 de Enero de 1900, pudiendo así conocer la fecha actual con asombrosa precisión.

Por suerte, en las librerías del Shield WiFi se incluye una librería que facilita la adquisición de la fecha por medio de NTP. Su implementación está en el Sketch del Arduino UNO, obteniéndose la hora, minutos, segundos, día del mes, mes y año de la medida.

La siguiente imagen muestra el esquema del "sketch" de la placa Arduino UNO:

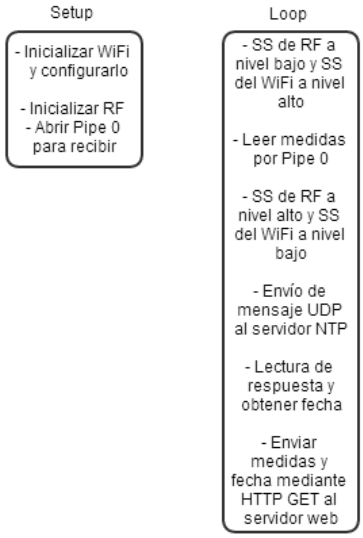


Figura 18. Sketch de la placa Arduino UNO Final

A la hora de subirlo al Servidor Web se decidió recurrir al esquema de:

HH MM SS DIA MES AÑO, para facilitar su visualización desde un navegador Web.

Se ha tenido que adaptar el código de "sensor.php", e incluir un programa para mostrarlo en "datos.html", al igual que para las medidas.

La imagen siguiente muestra un diagrama de bloques de la estructuración final del Servidor Web.

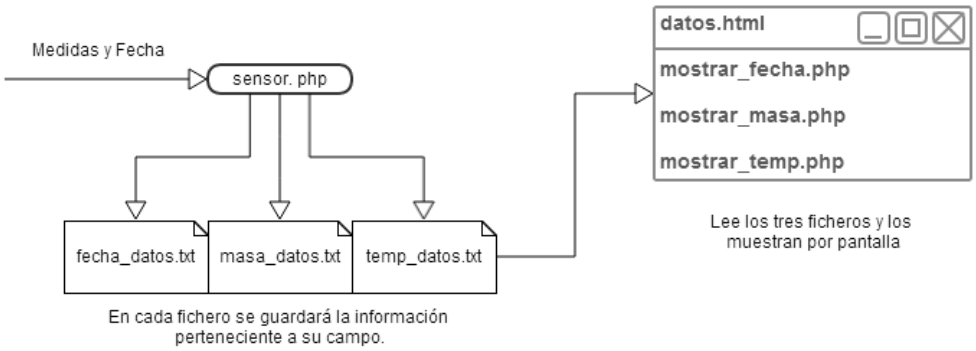


Figura 19. Diagrama del Servidor Web

En la App los cambios son más significativos, ya que primero hubo que ampliar el código de la petición para que realizase una más para obtener la fecha, y una vez obtenida con el formato indicado anteriormente, realizar la identificación de cada término.

Con todos los términos identificados, el último paso es el de por medio de la clase Calendar, nativa de Android, sacar los datos de fecha actuales y obtener la diferencia entre ambos.

Para su cálculo se usa la siguiente fórmula:

difhoras = (año_actual – *año_medida*) * 365 * 24 + (mes_actual- *mes_medida*) * dia_mes[mes_a_actual]* 24 + (dia_actual – *dia_medida*) * 24 + (hora_actual – *hora_medida*);

Siendo dia_mes un Array que contiene los días que tiene cada mes del año, para hacer una mejor aproximación de las horas de diferencia.

Esta fórmula nos da un resultado correcto para:

- Cambio de año.
- Cambio de mes.
- Cambio entre meses de 28,30 o 31 días.

Pero tiene los siguientes inconvenientes:

- No contempla los días exactos cuando hay una diferencia de dos meses o más.
- No contempla los años bisiestos.

Dado que tampoco se quiere un resultado con una gran exactitud, se optó por esta solución, debido a su sencillez al implementarse, y se añadió un TextView en el layout para mostrar al usuario este dato anteriormente calculado.

El esquema definitivo de la aplicación en Android quedaría de la siguiente manera:

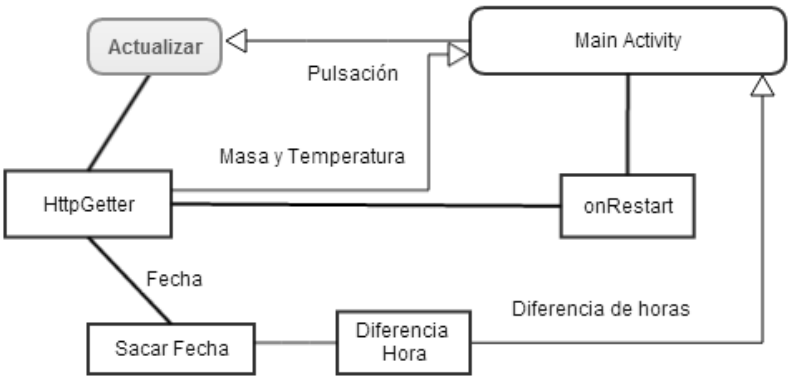


Figura 20. Diagrama de la aplicación en Android

Combinando todo lo antes mencionado, se puede extraer el diagrama de bloques del que sería el proyecto final que se plantea y que es objeto de este trabajo.

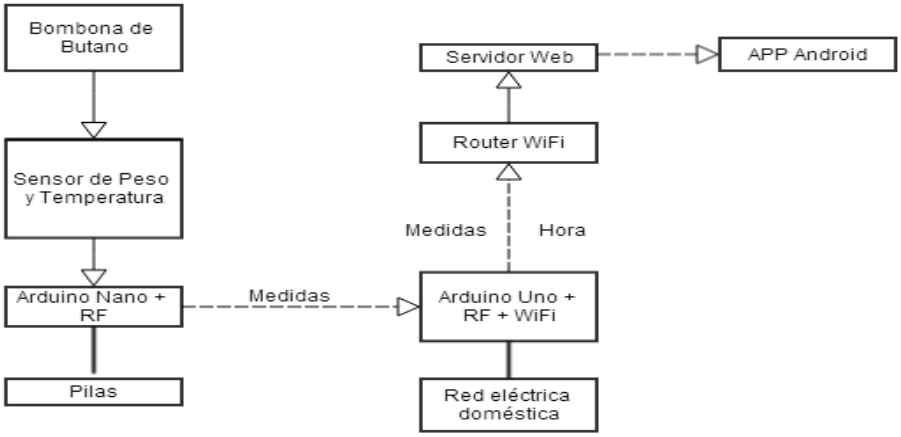
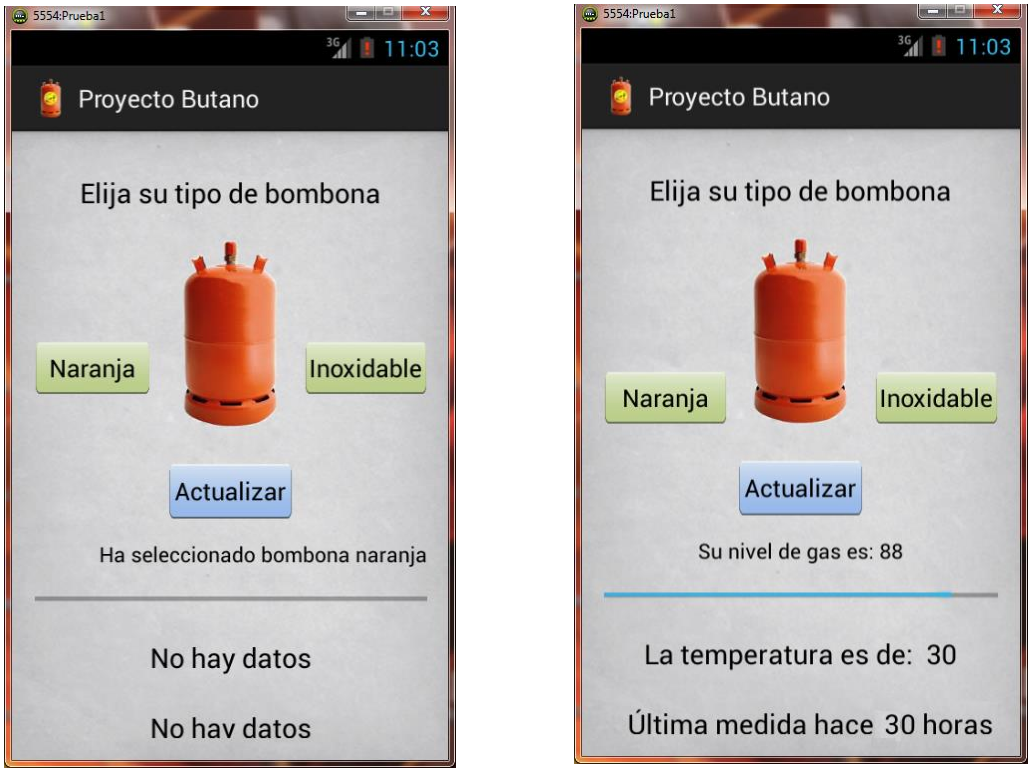


Figura 21. Diagrama del proyecto final

4.1.10 RESULTADO FINAL

Los sensores han sido diseñados, las conexiones con el Arduino NANO y su respectiva comunicación con el Arduino UNO probadas y verificadas. Se ha conseguido subir los datos de medidas y la fecha de dicha medida a un servidor y guardar los datos, para luego poder ser obtenidos por una Aplicación en Android y ser visualizados gráficamente por el usuario, los objetivos que se plantearon para este proyecto se han cumplido, tal como se muestra en las siguientes capturas de pantallas del terminal Android.



Figuras 22 y 23. Capturas de la aplicación en Android.

La primera imagen representa la aplicación una vez se elige la botella naranja, quedando a la esperan de la pulsación del botón "Actualizar" para realizar las peticiones GET y descargar los datos del servidor para ser mostrados al usuario.

Capítulo 5. PLIEGO DE CONDICIONES

5.1 ELECCIÓN DE MATERIAL Y EQUIPOS

Al plantearse el proyecto la primera decisión que se tomó en torno al material a utilizar fue los microcontroladores con los que se iba a trabajar.

Se escogió utilizar placas Arduino por su bajo coste, facilidad de programación y gran cantidad de soporte por parte de la comunidad en Internet, así como por las grandes variantes que ofrece en cuanto a comunicación con periféricos.

La placa Arduino NANO fue escogida por su bajo consumo y tamaño (18.5 mm x 43.2 mm), pudiendo ser colocada con facilidad en el soporte que se utilizaría como sensor de la masa del envase.

Para la placa externa se seleccionó el Arduino UNO, por ser un modelo estándar el cual cumplía con creces con las especificaciones que se pedían para ese sistema, además de tener un bajo coste.

A continuación de la selección de las placas Arduino el objetivo era la elección de todo el equipo para los sensores de masa y temperatura.

El sensor de temperatura como se ha explicado se construirá mediante un circuito integrado, el LM35, el cual incorpora todo lo necesario para funcionar, tan solo debe ser alimentado.

De la variedad de modelos en el mercado se escogió el LM35A, por tener un Offset 4 mV inferior a los otros modelos, siendo éste un parámetro muy interesante de reducir, pues introduce errores en la medida.
En el caso del sensor de masa, la primera elección fue de qué galgas extensiométricas se utilizarían, y tras una exhaustiva búsqueda las elegidas fueron el modelo 632-168 de la serie RS, las cuales cumplen con las especificaciones que se requerían, con un factor de galga ligeramente superior al estándar, una factor de deformación dentro de lo normal (10000 με) y una impedancia nominal de 120 Ω.
En lo concerniente al sensor de masa restan por seleccionar un circuito integrado que nos proporcione los 4 A.O. que son necesarios para el diseño del acondicionador. Se escogió el CI LM324, el cual incorpora los 4 A.O. necesarios, además de soportar alimentación unipolar, con la cual trabajará nuestro sensor. De los diferentes modelos de LM324, se escogió el LM324A por la reducción de la tensión de Offset máxima respecto a los otros modelos, parámetro muy importante por las características de la señal a tratar, que tendrá variaciones de mV una vez amplificada.

Para la conexión WiFi se barajó la posibilidad de comprar el Shield WiFi oficial de Arduino, pero debido a su alto coste, se decidió buscar opciones más económicas. Finalmente, tras comparar diferentes ofertas por Internet se escogió el siguiente módulo:

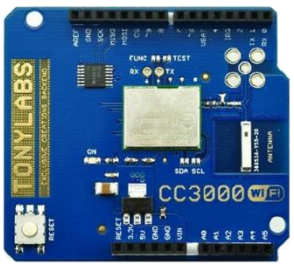


Figura 24. Shield WiFi

El cual incorpora un circuito integrado de Texas Instruments, el CC3000 que es el que proporciona la conectividad WiFi con su propia antena inalámbrica incorporada, y con todos los componentes necesarios para funcionar y pudiendo acoplarse perfectamente a la placa Arduino UNO.

Para la comunicación entre las dos placas Arduino la idea inicial fue usar módulos XBee, los cuales se basan en el protocolo ZigBee, pero tras una búsqueda en el mercado se vio que su precio era demasiado elevado para el presupuesto del proyecto, por lo que se tuvieron que barajar otras opciones más acordes a este presupuesto.

Se encontraron unos módulos (nRF24L01) los cuáles trabajan sobre el estándar 802.15.4 y cuyas especificaciones son muy parecidas a las ofrecidas por los módulos estándar de ZigBee, pero siendo su coste notoriamente inferior. Tras comprobar que se adaptaban a las necesidades del sistema se adquirió un par para ejercer de transmisor y receptor.

Para conectar los módulos con las placas Arduino fueron necesarios cables de cobre con cubierta de plástico para aislarlos, proporcionados por el Departamento de Ingeniería Electrónica de la Universidad Politécnica de Valencia, así como los componentes pasivos: resistencias, potenciómetros y el circuito integrado LM35A.

Para solventar el tema del servidor Web, como se ha comentado anteriormente, se optó por adquirir un hosting gratuito, pudiendo alojar en él todo lo necesario para el desarrollo del proyecto.

Para la prueba y ejecución de la aplicación de Android se usará un smartphone privado, siendo luego éste el del usuario propio.

5.2 ESQUEMAS

5.2.1 Sensor de Peso

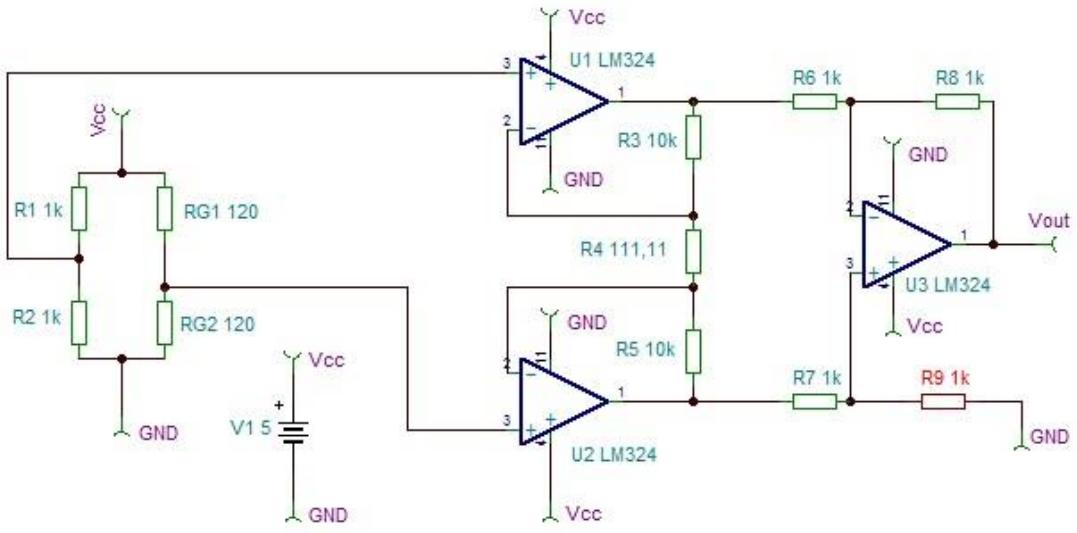


Figura 25. Esquema del sensor de peso

5.2.2 Divisor de Tensión

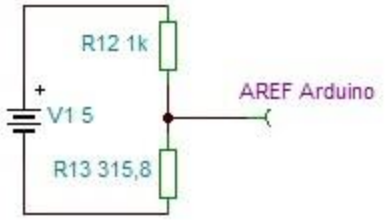


Figura 26. Esquema del divisor de tensión

5.2.3 Estructura de la lámina de flexión

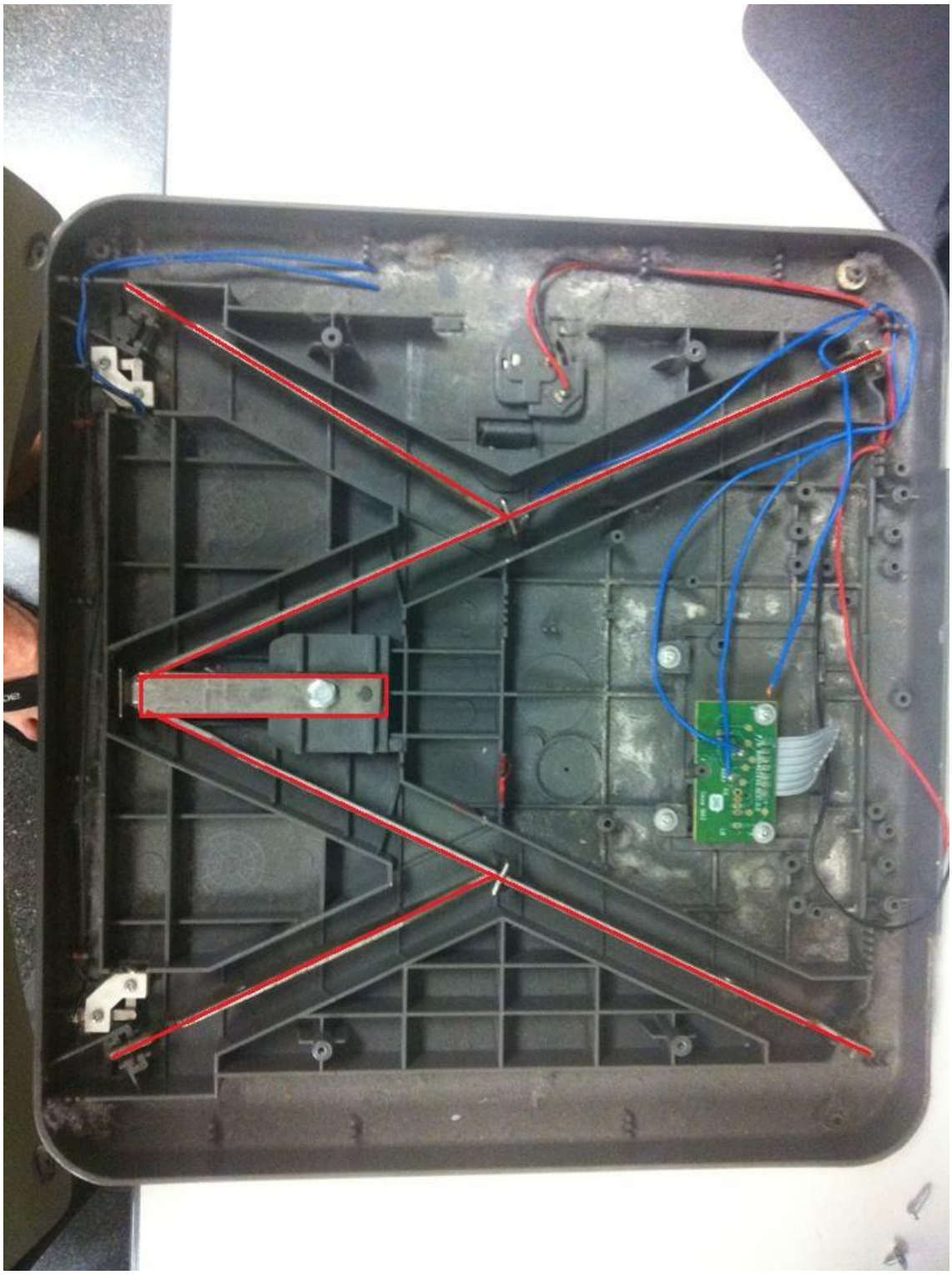


Figura 27. Estructura de la lámina de flexión

5.2.4 *Montaje real de las dos placas Arduinos con sus respectivos periféricos*

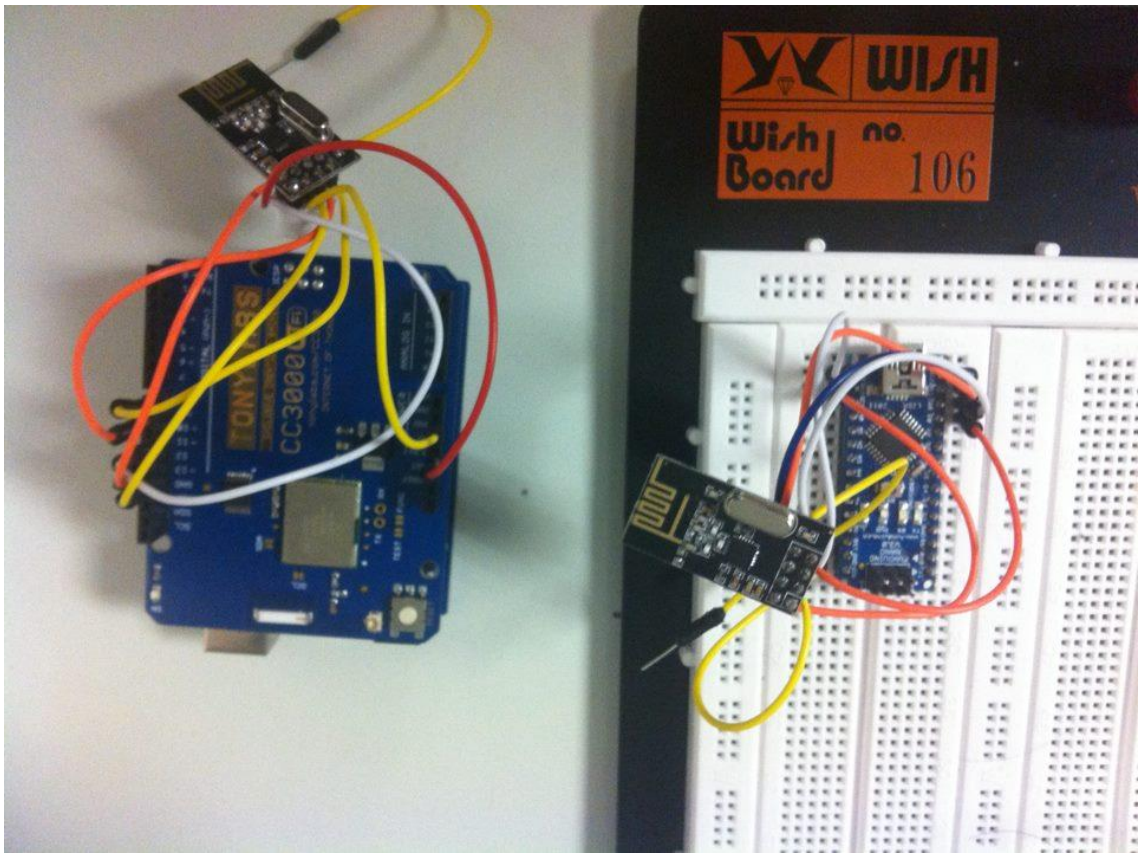


Figura 28. Montaje de ambas placas Arduino

5.2.5 *Arduino NANO conectado a nRF24L01*

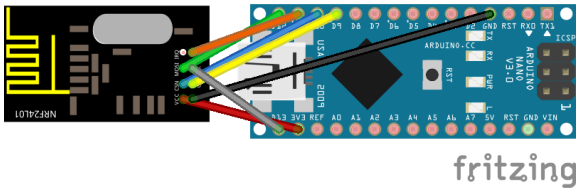


Figura 29. Esquema de las conexiones del Arduino NANO

Rojo → Vcc, Negro → GND

Azul → SS, Amarillo → CE

Verde → MISO, Gris → SCLK

Naranja → MOSI

5.2.6 *Arduino UNO conectado a nRF24L01 y módulo WiFi*

No se ha usado para el esquemático un Shield debido a que el objetivo era mostrar la conexión de los pines que usa el módulo WiFi, por lo que se ha elegido un módulo WiFi externo con idénticos pines útiles, para poder representar esta conexión correctamente.

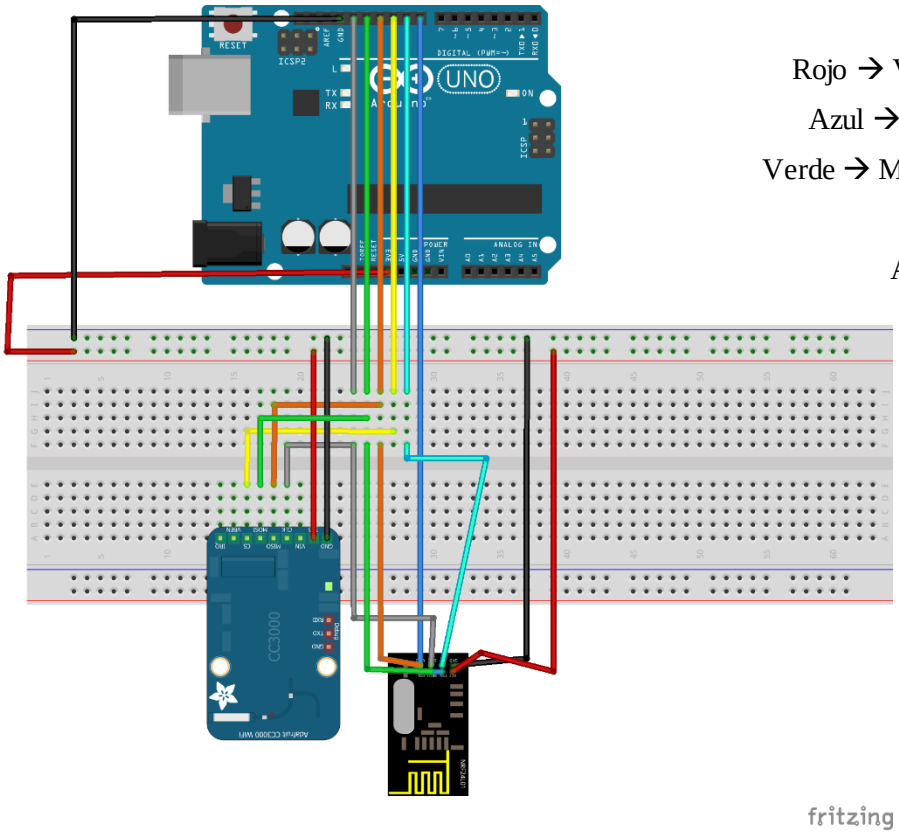


Figura 30. Esquema de conexiones del Arduino UNO

Capítulo 6. CONCLUSIÓN Y PROPUESTA DE TRABAJO FUTURO

A lo largo del presente texto se ha expuesto la idea original del proyecto, el proceso de desarrollo a la que la sometió, su evolución y el resultado final que se consiguió. Cada una de las partes en las que se dividió el trabajo fue resulta satisfactoriamente, pudiendo incluso agregar nuevas ideas para mejorar los resultados obtenidos.

Este proyecto se enfocó como una aplicación muy concreta de un sistema que puede ser usado para multitud de cosas, pero se prefirió darle un enfoque práctico a un problema real. Cuando se comenzó a fraguar la idea de este trabajo, no se tenían conocimientos de programación en Android, ni se sabía nada de PHP o HTML, ni siquiera cómo comunicar una placa Arduino con un router WiFi, el resultado de todo este tiempo invertido en el desarrollo de este proyecto ha estado bien empleado, pues se han ganado muchos conocimientos y se han asentado otros muchos, pudiendo además haber mejorado las habilidades en cuanto a la gestión de un proyecto con diversas partes independientes y la necesidad de organizar cada una de ellas para que el conjunto funcione como se esperaba.

6.1 MEJORAS FUTURAS

Tal y como se comentó en el apartado de la medida de temperatura, a éste valor sensado se le podría dar uso si se hiciera un estudio empírico del nivel mínimo de gas según la temperatura ambiente para el cual el calentador puede prender y funcionar, pudiendo avisar al usuario en su smartphone con una notificación cuando se fuese a alcanzar dicho punto crítico.

También podría ampliarse el servicio del servidor Web, haciéndolo una base de datos con usuarios diferentes y que cada uno consultase desde su propio terminal Android sus medidas propias, con un identificador en las palcas Arduino que ayudara a esta organización.

Para que el sistema pudiese ser libremente distribuido por los hogares, se debería dotar a la placa Arduino NANO de una conexión a un puerto USB hembra, el cual realizará la lectura de un fichero en el cual iría la SSID de la red WiFi doméstica, el tipo de clave usada y la contraseña para acceder a la misma, dejando toda su configuración y programa de fábrica intacto.

Por último, se podría extrapolar este sistema a una red de sensores por el hogar que permitan al usuario tener un control más exhaustivo de lo que ocurre en su hogar.

Un ejemplo de esto sería usar diversos sensores de temperatura por el hogar, extrayendo una media de las medidas y que el usuario desde su smartphone pudiese elegir si encender o apagar la calefacción/aire acondicionado según sus necesidades. Para lograr esta tarea, se debería incluir un microcontrolador en el mando inalámbrico de dicho aire acondicionado o calefacción.

Todo esto son ideas sin desarrollar, las cuales deberían tratarse sin apresurarse en su desarrollo, sentando unas buenas bases y teniendo claro qué es lo que se pretende implementar y las limitaciones que se nos imponen.

Capítulo 7. BIBLIOGRAFÍA

LEGISLACIÓN

Real Decreto 919/2006: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2006-15345

Norma NTP 209:

http://www.insht.es/InshtWeb/Contenidos/Documentacion/FichasTecnicas/NTP/Ficheros/201a300/ntp_209.pdf

CIRCUITOS INTEGRADOS y SENSORES

Galgas Extensiométricas:

http://docs-europe.electrocomponents.com/webdocs/0077/0900766b80077de6.pdf

LM324: http://www.ti.com/lit/ds/symlink/lm324.pdf

LM35A: http://www.ti.com/lit/ds/symlink/lm35.pdf

PLACAS Y MÓDULOS ARDUINO:

UNO: http://arduino.cc/en/Main/arduinoBoardUno

NANO: http://arduino.cc/es/Main/ArduinoBoardNano

Shield WiFi: http://www.tonylabs.com/product/cc3000

nRF24L01:
http://www.nordicsemi.com/eng/content/download/2726/34069/file/nRF24L01P_Product_Specification_1_0.pdf

Tutoriales CC3000, NRF, WDT:

http://openhardware.pe/transceptores-nrf24l01-2-4ghz-radio-wireless-how-to/

http://bitknitting.wordpress.com/2013/09/16/test-send-data-to-a-home-web-server/

http://interface.khm.de/index.php/lab/experiments/sleep_watchdog_battery/

PROGRAMACIÓN

SPI: http://www.circuitsathome.com/mcu/running-multiple-slave-devices-on-arduino-spi-bus

PHP: http://www.webestilo.com/php/php09b.phtml

HTML: http://www.tutorialmonsters.com/introduccion-al-html/

http://www.tutorialmonsters.com/elementos-y-etiquetas-de-html/

http://www.tutorialmonsters.com/html-formato-basico-de-texto/

CSS: http://www.tutorialmonsters.com/fundamento-basico-de-css/

ANDROID: http://developer.android.com/

http://stackoverflow.com/questions/3505930/make-an-http-request-with-android

http://android-developers.blogspot.com.es/2011/09/androids-http-clients.html

http://www.wikipedia.org

Capítulo 8. ANEXOS

LIBRERÍAS ARDUINO

Se usarán dos conjuntos de librerías diferentes: unas para la configuración y control del módulo de RF y otras para Shield WiFi.

Shield WiFi: https://github.com/adafruit/Adafruit_CC3000_Library

Módulo nrf24l01: https://github.com/maniacbug/RF24/

Ambas están disponibles en GitHub y son de código abierto.

Para el Servidor Web se ha usado el fichero de JavaScript "jquery-2.0.3.min.js", descargado de la siguiente dirección: http://jquery.com/download/. También es de código abierto.

PROGRAMAS

SKETCH Arduino Nano

```
//Librerías del módulo de RF.
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"
#include "printf.h"
//Configuración del módulo y asignación de pines: el Slave Enable al pin 9 de la placa Arduino,
//y el Slave Select al pin 10.
#include <avr/sleep.h>
#include <avr/power.h>
#include <avr/wdt.h>
volatile int f_wdt = 1;

RF24 radio(9,10);

//Booleano que usaremos para enviar la medida de Temperatura o la de masa.

bool ping_pong = false;

//Búfer donde guardaremos los datos a enviar.
unsigned char data[3] = {0};

// Direcciones de las Pipes que usaremos para comunicarnos.
const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };

void setup(void)
```

```
{
//Inicialización del puerto serie (para ver los datos en pantalla), y del módulo RF.
  Serial.begin(115200);
  printf_begin();
  radio.begin();


  // Especificamos el número de veces que intentará transmitir hasta que de errónea la
//comunicación.
  radio.setRetries(15,15);
//Hay que empezar la escucha para luego detenerla y así poder enviar datos satisfactoriamente.
  radio.startListening();
  radio.stopListening();


//Definimos las pipes que usará el módulo para comunicarse.
  radio.openWritingPipe(pipes[0]);
  radio.openReadingPipe(1,pipes[1]);
  radio.printDetails();


  /* Configuración del Temporizador Watchdog*/


  MCUSR &= ~(1<<WDRF); //Limpia el flag de reset
  WDTCSR |= (1<<WDCE) | (1<<WDE); //selecciona el preescalado del reloj
  WDTCSR = 1<<WDP0 | 1<<WDP3; //Se configura el Temporizador a 8 segundos
  WDTCSR != _BV(WDIE); //Habilita la interrupción del Watchdog


}


void loop(void)
{
    //Generamos las medidas ficticias y según toque el turno enviará una u otra.
    //Antes de enviar el dato, pondremos 'P' o 'T' como primer carácter del búfer para que el
//receptor sepa qué dato le estamos pasando.


  if(f_wdt == 1){
    int medidamasa = analogRead(1);
    int medidaTemp = analogRead(2);


    Serial.println("Enviando...");
```

```
    if(ping_pong==true){
      printf("Enviando masa: %d\r\n",medidamasa);
      data[0]='P';
      data[1] = medidamasa >> 8;
      data[2] = medidamasa;
      bool ok = radio.write(data, sizeof(int)+1);
      if (ok)
        printf("ok\n\r");
      else
        printf("fallo\n\r");
//De nuevo se debe poner en escucha y pararlo para poder transmitir de nuevo, ya que por //defecto espera una respuesta que no se usará.
        radio.startListening();
        radio.stopListening();
        ping_pong=true;
    }
    else{
      printf("Enviando Temperatura: %d\r\n",medidamasa);

      data[0]='T';
      data[1] = medidaTemp >> 8;
      data[2] = medidaTemp;
      bool ok = radio.write(data, sizeof(int)+1);
      if (ok)
        printf("ok\n\r");
      else
        printf("fallo\n\r");
        radio.startListening();
        radio.stopListening();
        ping_pong=true;

    }
    f_wdt=0;
    enterSleep();
  }
  else{
  }
}
```

```
ISR(WDT_vect){
 if(f_wdt == 0)f_wdt=1;
}

void enterSleep(void){
 set_sleep_mode(SLEEP_MODE_PWR_DOWN);
 sleep_enable();
 sleep_mode();
 /* Volverá aquí al salir del modo sleep */
 sleep_disable();
 power_all_enable();
}
```

SKETCH Arduino Uno

```
//Librerías del módulo RF.
#include "nRF24L01.h"
#include "RF24.h"
#include "printf.h"

//Librerías del Shield WiFi.
#include <Adafruit_CC3000.h>
#include <ccspi.h>
#include <SPI.h>
#include <string.h>
#include "utility/sntp.h"


//Definimos los pines que usará el Shield WiFi.
#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  5
#define ADAFRUIT_CC3000_CS    10
//Generación del objeto que controla el Shield WiFi.
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS,
ADAFRUIT_CC3000_IRQ, ADAFRUIT_CC3000_VBAT,
                         SPI_CLOCK_DIVIDER);
```

```
//Nombre y contraseña de la red WiFi doméstica a la que estará conectado.
#define WLAN_SSID       "Orange-4963"
#define WLAN_PASS       "C7F45444"
#define WLAN_SECURITY   WLAN_SEC_WPA2

//Nombre de la página web a la que se accederá.
#define WEBSITE      "proyectoteleco.esy.es"
#define WEBPAGE      "/medidas/sensor.php"

uint32_t ip;

//Configuración del Módulo RF y creación de las Pipes a usar.
RF24 radio(9,8);
const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };

//Se define el Servidor NTP a usar, así como la zona horaria, eligiendo en este caso España con Horario de Verano.
sntp mysntp = sntp(NULL, "time.nist.gov", (short)(1 * 60), (short)(2 * 60), true);

//Tiempo transcurrido desde 1/1/1900
SNTP_Timestamp_t now;

//Objeto que genera la librería sntp.h que proporciona facilidades para conocer la fecha.
NetTime_t timeExtract;

int horas,minutos,segundos,dia,year;
String mes;

void setup(void)
{

  pinMode(8,OUTPUT); //SS del Módulo RF.
  pinMode(10,OUTPUT); //SS del Shield WiFi.
  Serial.begin(115200);



  // Inicialización del Shield WiFi.
  Serial.println(F("\nInitializing..."));
```

```
if (!cc3000.begin())
{
  Serial.println(F("Couldn't  begin()! Check your wiring?"));
  while(1);
}

//Se conectará a la red WiFi definida.
Serial.print(F("\nAttempting  to connect to ")); Serial.println(WLAN_SSID);
if (!cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY)) {
  Serial.println(F("Failed!"));
  while(1);
}

Serial.println(F("Connected!"));

//Petición DHCP
Serial.println(F("Request  DHCP"));
while (!cc3000.checkDHCP())
{
  delay(100);
}

ip = 0;
//Búsqueda de la dirección IP de la página a la que se quiere acceder.
Serial.print(WEBSITE); Serial.print(F("  -> "));
while (ip == 0) {
  if (! cc3000.getHostByName(WEBSITE, &ip)) {
    Serial.println(F("Couldn't  resolve!"));
  }
  delay(500);
}

cc3000.printIPdotsRev(ip);

//Inicialización  y configuración del Módulo RF.
radio.begin();
printf_begin();
```

```
  printf("\n\rRF24/examples/GettingStarted/\n\r");
  radio.setRetries(15,15);
  radio.startListening();
  //Definimos las Pipes a usar, en este caso las contrarias a las del transmisor.
  radio.openWritingPipe(pipes[1]);
  radio.openReadingPipe(1,pipes[0]);
  radio.printDetails();
}

void loop(void)
{
  int medidaMasa = 0;
  int medidaTemp = 0;

  unsigned char data[3] = {0}; //Búfer donde se guardarán los datos recibidos.

  //Le damos comunicación al módulo de RF y se la quitamos al Shield WiFi.
  digitalWrite(8,LOW);
  digitalWrite(10,HIGH);
  if ( radio.available() )
      {
       bool done = false; //boolean que controlará cuando se reciben datos.
      //Estos dos booleans controlarán cuando se haya recibido la medida de Masa y la de
Temperatura.
       bool masa_recibida=false;
       bool temp_recibida=false;
      while (!done)
      {
       done = radio.read(data, sizeof(int)+1);
       if(data[0] == 'M'){
         medidaMasa = data[2] | data[1] << 8;
         printf("Masa: $d\r\n",medidaMasa);
         masa_recibida=true;
       }
       if(data[0] == 'T'){
         medidaTemp = data[2] | data[1] << 8;
         printf("Tempeatura: %d\r\n",medidaTemp);
         temp_recibida=true;
```

```
        }
            delay(20);
       }
  }
if(temp_recibida && masa_recibida){
  //Cortamos la comunicación con el módulo RF y la habilitamos para el Shield WiFi.
  digitalWrite(8,HIGH);
  digitalWrite(10,LOW);
  SubirDatos(medidaMasa,medidaTemp); //subiremos las medidas al Servidor Web.
  delay(10000); //esto se podría mejorar con un control del tiempo más exaustivo, para reducir el consumo.
}
}
 void SubirDatos(int Masa, int Temp){

  obtenerNTP(); //Esta función obtiene la fecha del Servidor NTP.
  String fecha = String(horas) + " " + String(minutos) + " " + String(segundos)
  + " " + String(dia) + " " + mes + " " + String(year);

  //Creamos las Strings de petición HTTP GET y lo pasamos a cadena de carácteres.
  String Smasa ="?masa="+String(Masa);
  String Stemp ="&temp="+String(Temp);
  String Sfecha ="&fecha="+fecha;
  char cmasa [20];
  char ctemp [20];
  char cfecha [30];
  Smasa.toCharArray(cmasa,Smasa.length()+1);
  Stemp.toCharArray(ctemp,Stemp.length()+1);
  Sfecha.toCharArray(cfecha,Sfecha.length()+1);

  //Habilitamos una conexión TCP con nuestro Servidor Web y enviamos una petición GET con los valores de medidas y fecha.
  Adafruit_CC3000_Client www = cc3000.connectTCP(ip, 80);
  if (www.connected()) {
    www.fastrprint(F("GET "));
    www.fastrprint(WEBPAGE);
    www.fastrprint(cmasa);
    www.fastrprint(ctemp);
```

```
    www.fastrprint(cfecha);
    www.fastrprint(F(" HTTP/1.1\r\n"));
    www.fastrprint(F("Host: ")); www.fastrprint(WEBSITE); www.fastrprint(F("\r\n"));
    www.fastrprint(F("\r\n"));
    www.println();
  } else {
    Serial.println(F("Connection  failed"));
    return;
  }
// Leemos los datos recibidos hasta que la conexión se cierra
  unsigned long lastRead = millis();
  while (www.connected()){
    while (www.available()) {
      char c = www.read();
      Serial.print(c);
      lastRead = millis();
    }
  }
  www.close(); //Se cierra la conexión TCP con el Servidor.
  }

  void obtenerNTP(void){

  mysntp.UpdateNTPTime(); //Envio de mensaje UDP al Servidor NTP.
  mysntp.ExtractNTPTime(mysntp.NTPGetTime(&now, true), &timeExtract);

  //Del objeto timeExtract obtenedremos la hora,minutos,segundos,día  del mes, mes y año de la
medida.
  horas = timeExtract.hour;
  minutos = timeExtract.min;
  segundos = timeExtract.sec;
  dia = timeExtract.mday;
  switch(timeExtract.mon){
  case 0:mes = "Enero"; break;
  case 1:mes = "Febrero"; break;
  case 2:mes = "Marzo"; break;
  case 3:mes = "Abril";  break;
  case 4:mes = "Mayo"; break;
```

```
case 5:mes = "Junio"; break;
case 6:mes = "Julio"; break;
case 7:mes = "Agosto"; break;
case 8:mes = "Septiembre"; break;
case 9:mes = "Octubre"; break;
case 10:mes = "Noviembre"; break;
case 11:mes = "Diciembre"; break;
}
year = timeExtract.year;
}
```

Servidor Web

Datos.html

Este programa nos permitirá visualizar los datos guardados de las medidas realizadas y de su fecha de realización.

```
<!DOCTYPE html>
<html>
<head>
	<script src="jquery-2.0.3.min.js"></script>
	<link rel="stylesheet" type="text/css" href="style.css">
	<title>Proyecto Butano</title>
</head>

<body>
	<div class="data">
		<div class="dataTitle">Temperatura: </div>
		<div id="Display_temp">Cargando...</div>
	</div>

	<div class="data">
		<div class="dataTitle">Masa: </div>
		<div id="Display_masa">Cargando...</div>
	</div>

	<div class="data">
```

```
            <div class="dataTitle">Fecha: </div>
            <div id="Display_fecha">Cargando...</div>
        </div>

        <script type="text/javascript">

            setInterval(function()
        {
            $("#Display_temp").load('mostrar_temp.php');
            $("#Display_masa").load('mostrar_masa.php');
            $("#Display_fecha").load('mostrar_fecha.php');
        }, 1000);

        </script>
</body>
</html>
```

Style.css

Un estilo sencillo para mejorar lo mostrado con el programa anterior.

```
.data {
        font-size: 36px;
        margin: 30px;
        margin-left: 30px;
        clear: both;
}

.dataTitle {
        float: left;
        margin-right: 20px;
}

body {
        font-family: Helvetica;
}
```

Los tres códigos siguientes tienen el mismo fin: acceder a los ficheros correspondientes a cada dato y leer la información para mostrarla por pantalla.

Mostrar_fecha.php

```
<?php
        $fichero = "fecha_datos.txt";
        $f = fopen($fichero, 'r');
        $linea  = fgets($f);
        fclose($f);
        echo $linea;
?>
```

Mostrar_temp.php

```
<?php
        $fichero = "temp_datos.txt";
        $f = fopen($fichero, 'r');
        $linea  = fgets($f);
        fclose($f);
        echo "$linea ºC";
?>
```

Mostrar_masa.php

```
<?php
        $fichero = "masa_datos.txt";
        $f = fopen($fichero, 'r');
        $linea  = fgets($f);
        fclose($f);
        echo "$linea kg";
?>
```

Sensor.php

Guarda los datos que llegan mediante HTTP GET en sus correspondientes ficheros.

```
<?php
```

```
    if ($_GET["temp"] && $_GET["masa"] && $_GET["fecha"]{

            $fichero = "temp_datos.txt";
                    $f = fopen($fichero, 'w');
                    fwrite($f, $_GET["temp"]);
                    fclose($f);

            $fichero = "masa_datos.txt";
                    $f = fopen($fichero, 'w');
                    fwrite($f, $_GET["masa"]);
                    fclose($f);

               $fichero = "fecha_datos.txt";
                    $f = fopen($fichero, 'w');
                    fwrite($f, $_GET["fecha"]);
                    fclose($f);
    }
?>
```

Aplicación Android

MainActivity.java

```
package com.upv.proyectobutano;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Calendar;
import java.util.StringTokenizer;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.StatusLine;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpUriRequest;
import org.apache.http.impl.client.DefaultHttpClient;
import android.app.Activity;
```

```
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.ProgressBar;
import android.widget.TextView;

public class MainActivity extends Activity {
        // Variable donde se guardará la medida de Masa
        public static int medidamasa = 0;
        // Variable donde se guardará la medida de Temperatura.
        public static int medidatemp = 0;
        /* Variable que controlará el tipo de bombona seleccionada.
         * 1 -> bombona naranja, 2 -> bombona de aluminio  */
        public static int bombona = 0;
        /* Variable que contendrá la diferencia de horas entre la última medida y
         *  la hora actual. */
        public static int diffhoras = 0;
        // Creamos una ProgressBar para mostrar el porcentaje de gas.
        public static ProgressBar barra;
        // Definimos su máximo en 100 para mostrarlo en %.
        private static int VALOR_MAX = 100;
        // String donde guardaremos la respuesta del Servidor.
        public static String mensaje = "";
        //Direcciones de los ficheros a los que se accederá
        public String URLtemp = "http://proyectoteleco.esy.es/medidas/temp_datos.txt";
        public String URLmasa = "http://proyectoteleco.esy.es/medidas/masa_datos.txt";
        public String URLfecha = "http://proyectoteleco.esy.es/medidas/fecha_datos.txt";

        /* Con este Array de int controlaremos los días de cada mes, para hacer un
         * mejor ajuste de la diferencia de horas */
        public int dia_mes[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
        public static int hora;
        public static int dia;
        public static int mes;
```

```
public static int year;

@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        barra = (ProgressBar) findViewById(R.id.progressBar1);
        barra.setMax(VALOR_MAX);
        // Localizamos el botón naranja en la vista
        final Button naranja = (Button) findViewById(R.id.naranja);
        // Creamos el Listener para dicho botón
        naranja.setOnClickListener(new Button.OnClickListener() {
                public void onClick(View arg0) {
                        // Cambiamos la imagen por la de la bombona naranja.
                        ImageView naranja_img = (ImageView)
findViewById(R.id.imagen1);
                        naranja_img.setImageResource(R.drawable.normal);
                        /* Cambiamos la variable que nos indica qué bombona está
                         * seleccionada. */
                        bombona = 1;
                        TextView texto = (TextView) findViewById(R.id.text_masa);
                        texto.setText("Ha seleccionado bombona Naranja");
                }
        });
        // Localizamos el botón aluminio  en la vista
        final Button inoxidable  = (Button) findViewById(R.id.inoxidable);
        // Creamos el Listener para dicho botón
        inoxidable.setOnClickListener(new Button.OnClickListener() {
                public void onClick(View arg0) {
                // Cambiamos la imagen por la de la bombona de aluminio.
                        ImageView inoxidable_img = (ImageView)
findViewById(R.id.imagen1);
                inoxidable_img.setImageResource(R.drawable.inoxidable);
                /* Cambiamos la variable que nos indica qué bombona está
                 * seleccionada. */
                bombona = 2;
                TextView texto = (TextView) findViewById(R.id.text_masa);
                texto.setText("Ha seleccionado bombona de Acero Inoxidable ");
```

```
                }
            });
            // Localizamos el botón de actualizar en la vista
            final Button miboton3 = (Button) findViewById(R.id.actualizar);
            // Creamos el Listener para dicho botón
            miboton3.setOnClickListener(new Button.OnClickListener() {
                public void onClick(View arg0) {
                    // Se crea una AsyncTask nueva para hacer la petición GET
                    new HttpGetter().execute(URLmasa, URLtemp, URLfecha);
                    float valor = 0;
                    TextView texto = (TextView) findViewById(R.id.text_masa);
                    TextView textotemp = (TextView) findViewById(R.id.text_temp);
                    TextView textohora = (TextView) findViewById(R.id.text_hora);
                    /* Dependiendo del valor de bombona haremos una operación u
                     * otra, como fue detallado en su apartado correspondiente.
                     */
                    if (bombona == 1)
                        valor = (medidamasa - 139) / 1.25f;
                    if (bombona == 2)
                        valor = (medidamasa - 55) / 1.25f;
                    // Sólo entrará cuando se haya seleccionado bombona.
                    if (bombona != 0) {
                        medidamasa = (int) valor;
                        texto.setText("Su nivel de gas es: "
                                + Integer.toString(medidamasa));
                        barra.setProgress(medidamasa);
                        textotemp.setText("La temperatura es de: "
                                + Integer.toString(medidatemp));
                        textohora.setText("Última medida hace "
                                + Integer.toString(diffhoras) + "horas");
                    }
                }
            });
        }
        /* Al implementar este método se logra que cada vez que se vuelva
         * a abrir la App se realice una nueva solicitud */
```

```
@Override
protected void onRestart() {
        super.onRestart();
        new HttpGetter().execute(URLmasa, URLtemp, URLfecha);
}
/* Se define la Clase HttpGetter, siendo esta una AsyncTask que controlará
 * la solicitud HTTP GET al Servidor y la interpretación del contenido de la
 * respuesta */
class HttpGetter extends AsyncTask<String, Void, Void> {
        @Override
        protected Void doInBackground(String... urls) {
                // Se crean las variables necesarias para la comunicación.
                StringBuilder builder = new StringBuilder();
                HttpClient client = new DefaultHttpClient();
                HttpGet httpGet = new HttpGet(urls[0]);
                try {
                        // Ejecutamos la petición y obtenemos la respuesta del Servidor.
                        HttpResponse response = client
                                        .execute((HttpUriRequest) httpGet);
                        StatusLine statusLine = response.getStatusLine();
                        int statusCode = statusLine.getStatusCode();
                        // El código 200 indica que la petición ha sido correcta.
                        if (statusCode == 200) {
                                /* Se extraen los datos de interés contenidos en el
                                 * paquete de respuesta. */
                                HttpEntity entity = response.getEntity();
                                InputStream content = entity.getContent();
                                BufferedReader reader = new BufferedReader(
                                                new InputStreamReader(content));
                                String line;
                                /* Se construye un StringBuilder con cada línea
                                 * del mensaje. */
                                while ((line = reader.readLine()) != null) {
                                        builder.append(line);
                                }
                                Log.v("Getter", "Datos Masa: " + builder.toString());
                                /* El StringBuilder se transforma en una String para
```

```
                 * obtener los valores. */
                mensaje = builder.toString();
                medidamasa = Integer.parseInt(mensaje);
        } else {
                Log.e("Getter", "Fallo en Masa");
        }
} catch (ClientProtocolException e) {
        e.printStackTrace();
} catch (IOException e) {
        e.printStackTrace();
}
builder = new StringBuilder();
//Se repite el mecanismo anterior, pero para la siguiente dirección.
client = new DefaultHttpClient();
httpGet = new HttpGet(urls[1]);
try {
        HttpResponse response = client
                        .execute((HttpUriRequest) httpGet);
        StatusLine statusLine = response.getStatusLine();
        int statusCode = statusLine.getStatusCode();
        if (statusCode == 200) {
                HttpEntity entity = response.getEntity();
                InputStream content = entity.getContent();
                BufferedReader reader = new BufferedReader(
                                new InputStreamReader(content));
                String line;
                while ((line = reader.readLine()) != null) {
                        builder.append(line);
                }
                Log.v("Getter", "Datos Temperatura: " +
                builder.toString());
                mensaje = builder.toString();
                medidatemp = Integer.parseInt(mensaje);
        } else {
                Log.e("Getter", "Fallo en Temperatura");
        }
} catch (ClientProtocolException e) {
```

```
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        /* De nuevo se volverá a seguir el mismo algoritmo para la última
         * dirección. */
        builder = new StringBuilder();
        client = new DefaultHttpClient();
        httpGet = new HttpGet(urls[2]);
        try {
            HttpResponse response = client
                    .execute((HttpUriRequest) httpGet);
            StatusLine statusLine = response.getStatusLine();
            int statusCode = statusLine.getStatusCode();
            if (statusCode == 200) {
                HttpEntity entity = response.getEntity();
                InputStream content = entity.getContent();
                BufferedReader reader = new BufferedReader(
                        new InputStreamReader(content));
                String line;
                while ((line = reader.readLine()) != null) {
                    builder.append(line);
                }
                Log.v("Getter", "Datos Fecha: " + builder.toString());
                mensaje = builder.toString();
                /* En este caso, no basta con convertirla respuesta,
                 * se debe analizar y extraer de ella los datos  */
                sacarFecha(mensaje);
            } else {
                Log.e("Getter", "Fallo en Fecha");
            }
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
```

```
}
/* Este método extraerá los diferentes campos de la respuesta de la petición
 * GET para conocer la fecha de la última medida. Se ha decidido prescindir
 * de los datos de minutos y segundos para el cálculo de la diferencia de
 * horas, por lo que no serán de interés el extraer dichos datos. */
void sacarFecha(String mensaje) {
        /* Se divide la String del mensaje en partes separadas por
         * los espacios en blanco. */
        StringTokenizer st = new StringTokenizer(mensaje);
        String TokenActual = new String();
        String Shora = new String();
        String Sdia = new String();
        String Smes = new String();
        String Syear = new String();

        Boolean bienhora = false;
        Boolean bienmin = false;
        Boolean biensec = false;
        Boolean biendia = false;
        Boolean bienmes = false;
        Boolean bienyear = false;
        /* Se analizará cada parte por separado, siguiendo el esquema de
         * HH MM SS DIA MES AÑO */
        while (st.hasMoreTokens()) {
                TokenActual = st.nextToken();
                Log.d("", "Este es el token:" + TokenActual);
                if (!bienhora) {
                        bienhora = true;
                        Shora = TokenActual;
                } else {
                        if (!bienmin) {
                                bienmin = true;
                        } else {
                                if (!biensec) {
                                        biensec = true;
                                } else {
                                        if (!biendia) {
```

```
                                        biendia = true;
                                        Sdia = TokenActual;
                                } else {
                                        if (!bienmes) {
                                                bienmes = true;
                                                Smes = TokenActual;
                                        } else {
                                                if (!bienyear) {
                                                        bienyear = true;
                                                        Syear = TokenActual;
                                                }
                                        }
                                }
                        }
                }
        }
}
// Quitamos los posibles espacios en blanco de las Strings
Shora = Shora.trim();
Sdia = Sdia.trim();
Smes = Smes.trim();
Syear = Syear.trim();

hora = Integer.parseInt(Shora);
dia = Integer.parseInt(Sdia);
year = Integer.parseInt(Syear);
/* Con este algoritmo se pasará del mes en String a su
 * correspondiente número. */
if (Smes.compareTo("Enero") == 0)
        mes = 1;
else {
        if (Smes.compareTo("Febrero") == 0)
                mes = 2;
        else {
                if (Smes.compareTo("Marzo") == 0)
                        mes = 3;
                else {
```

```
                                        if (Smes.compareTo("Abril") == 0)
                                                mes = 4;
                                        else {
                                                if (Smes.compareTo("Mayo") == 0)
                                                        mes = 5;
                                                else {
                                                        if (Smes.compareTo("Junio") == 0)
                                                                mes = 6;
                                                        else {
                                                                if (Smes.compareTo("Julio") ==
0)

                                                                        mes = 7;
                                                                else {
                                                                        if
(Smes.compareTo("Agosto") == 0)
                                                                                mes = 8;
                                                                        else {
                                                                                if
(Smes.compareTo("Septiembre") == 0)
                                                                                        mes    =
9;
                                                                                else {
                                                                                        if
(Smes.compareTo("Octubre") == 0)

        mes = 10;

                                                                                        else {

        if (Smes.compareTo("Noviembre") == 0)

        mes = 11;

        else if (Smes

                .compareTo("Diciembre") == 0)

        mes = 12;
                                                                                        }
                                                                                }
                                                                        }
                                                                }
```

```
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                                /*Se hace la llamda al método de cálculo de la diferencia de horas. */
                                diffhora();
                        }
                        /* Con esto método se consigue obtener la diferencia de horas entre la fecha
                         * actual y la fecha en la que se realizó la medida */
                        void diffhora() {
                                Calendar cal = Calendar.getInstance();
                                int hora_a = cal.get(Calendar.HOUR_OF_DAY);
                                int dia_a = cal.get(Calendar.DAY_OF_MONTH);
                                int mes_a = cal.get(Calendar.MONTH) + 1;
                                int year_a = cal.get(Calendar.YEAR);

                                diffhoras = (year_a - year) * 365 * 24 + (mes_a - mes) * dia_mes[mes_a]
                                                * 24 + (dia_a - dia) * 24 + (hora_a - hora);
                        }
} /* Final de MainActivity */
```

Activity_main.xml (Layout)

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/background"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >
    <Button
        android:id="@+id/inoxidable"
```

```
    android:layout_width="90dp"
    android:layout_height="40dp"
    android:layout_alignBaseline="@+id/naranja"
    android:layout_alignBottom="@+id/naranja"
    android:layout_alignParentRight="true"
    android:background="@drawable/boton_verde1"
    android:text="@string/inoxidable" />
<TextView
    android:id="@+id/text_temp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/text_masa "
    android:layout_centerHorizontal="true"
    android:layout_marginTop="51dp"
    android:text="@string/no_data"
    android:textColor="#000000"
    android:textSize="20sp" />

<TextView
    android:id="@+id/text_hora"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/text_temp"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="24dp"
    android:text="@string/no_data"
    android:textColor="#000000"
    android:textSize="20sp" />
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="15dp"
    android:text="@string/elija_su_bombona"
    android:textColor="#000000"
```

```
    android:textSize="20sp" />
<ProgressBar
    android:id="@+id/progressBar1"
    style="?android:attr/progressBarStyleHorizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/text_masa "
    android:layout_centerHorizontal="true"
    android:layout_marginTop="14dp" />
<TextView
    android:id="@+id/text_masa"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignRight="@+id/inoxidable"
    android:layout_below="@+id/actualizar"
    android:layout_marginTop="15dp"
    android:text="@string/no_seleccion"
    android:textColor="#000000"
    android:textSize="15sp" />
<Button
    android:id="@+id/naranja"
    android:layout_width="90dp"
    android:layout_height="40dp"
    android:layout_alignBottom="@+id/imagen1"
    android:layout_marginBottom="24dp"
    android:layout_toLeftOf="@+id/text_temp"
    android:background="@drawable/boton_verde1"
    android:text="@string/naranja" />
<ImageView
    android:id="@+id/imagen1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView1"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="20dp"
    android:src="@drawable/nada" />
<Button
```

```
        android:id="@+id/actualizar"
        android:layout_width="90dp"
        android:layout_height="40dp"
        android:layout_below="@+id/imagen1"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="26dp"
        android:background="@drawable/boton_azul"
        android:text="@string/actualizar" />
</RelativeLayout>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.upv.proyectobutano"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="9"
        android:targetSdkVersion="17" />
<uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.upv.proyectobutano.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Proyecto Butano</string>
    <string name="action_settings">Settings</string>
    <string name="elija_su_bombona">Elija su tipo de bombona</string>
    <string name="no_seleccion">No ha seleccionado el tipo de bombona</string>
    <string name="naranja">Naranja</string>
    <string name="inoxidable">Inoxidable</string>
    <string name="actualizar">Actualizar</string>
    <string name="no_data">No hay datos</string>
</resources>
```