UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

PROYECTO FINAL DE CARRERA

Desarrollo y mejoramiento de un dispositivo capaz de monitorear niveles de ozono troposférico

Titulación:
Ingeniería de Telecomunicaciones

Autor:
Sebastian Martínez Basurto
Director:
Jose María Barceló Ordinas

Campus Nord
Barcelona, España
Julio, 2018

Resumen

En un mundo en el que las circunstancias políticas, económicas, sociales y culturales, demanden un mayor conocimiento y conciencia de la realidad circundante, se torna necesaria una cultura y educación ambiental que propicie más espacios de acción y reflexión entre los ciudadanos. Justamente hacia esa directriz está inclinado el presente proyecto, al contribuir con el desarrollo de un nodo sensor capaz de registrar niveles de contaminación ambiental. El objetivo principal, es realizar una actualización hardware al nodo CAPTOR, con la finalidad de obtener un dispositivo preciso, sólido y, sobre todo, perdurable. Para poder lograrlo, se hace una investigación exhaustiva sobre las características, funcionalidades y cualidades del nuevo procesador Raspberry Pi. La actualización se despliega sin errores aparentes y muestra potencial para adquirir posibles mejoras en un futuro cercano.

Abstract

In a world in which political, economic, social and cultural circumstances demand a bigger knowledge and awareness of the surrounding reality, it becomes necessary an environmental culture that encourages more spaces of action and reflection between citizens. It is precisely towards this goal that the present project is inclined, to contribute with the development of a sensor node capable of register the levels of environmental pollution. The main objective is to realize a hardware update to the captor node, to obtain a precise, solid and, especially, a lasting device.

In order to achieve this, an exhaustive investigation is made about the characteristics, functionalities and qualities of the new Raspberry Pi processor. The update is displayed without apparent errors and shows potential to gather possible improvements in a near future.

Resum

En un món en què les circumstàncies polítiques, econòmiques, socials i culturals demanden un major coneixement i consciència de la realitat circumdant, es torna necessària una cultura i educació ambiental que propiciï més espais d'acció i reflexió entre els ciutadans. Justament vers aquesta directriu s'inclina el present projecte, contribuint amb el desenvolupament d'un node sensor capaç de registrar nivells de contaminació ambiental. L'objectiu principal és realitzar una actualització hardware al node CAPTOR, amb la finalitat d'obtenir un dispositiu precís, sòlid i sobre tot, perdurable. Per poder aconseguir-lo, es fa una investigació exhaustiva sobre les característiques, funcionalitats i qualitats del nou processador Raspberry Pi. L'actualització es desplega sense errors aparents i mostra potencial per adquirir possibles millores en un futur proper.

Agradecimientos

Agradezco la oportunidad brindada por Jorge García Vidal y por Jose María Barceló Ordinas de formar parte de un proyecto internacional a favor del medio ambiente y agradezco la ayuda ofrecida por mis compañeros del grupo SANS.

Agradezco a mis padres, Víctor Manuel Martínez y Minerva Basurto, porque sin su apoyo no podría estar viviendo este sueño de estudiar en otro país.

Agradezco a mi novia, Melanie Abreu, por estar ahí cuando más lo necesito y por nunca dejar de creer en mí.

Índice

1
Introducción

1.1 Antecedentes y justificación del proyecto

Actualmente, la contaminación del aire es una de las problemáticas ambientales que más perjudican al planeta. El origen de este fenómeno se remonta en la Gran Bretaña del siglo XVII, cuna de la revolución industrial. La economía que hasta ese entonces se basaba únicamente en el trabajo manual, fue reemplazada y superada por la industria y las actividades manufactureras.

Sin embargo, las fábricas, para poder producir bienes de consumo en dicha época, requerían de considerables cantidades de energía que obtenían mediante la quema de combustibles fósiles, los cuales, al entrar en contacto con la luz solar, sufren una reacción fotoquímica, generando así el contaminante que se conoce como ozono troposférico.

Según el último informe de la Agencia Europea de Medio Ambiente sobre la calidad del aire en Europa, se registra al ozono troposférico (O3) como uno de los 3 contaminantes con mayor índice de toxicidad para el ser humano, acompañado del dióxido de nitrógeno (NO2) y las partículas en suspensión (PM). El ozono troposférico es un contaminante secundario, ya que no es emitido a la atmosfera directamente, como se mencionó anteriormente, el O3 se produce cuando coexisten compuestos orgánicos volátiles no metánicos (COVNM), óxido de nitrógeno (NO), monóxido de carbono (CO) y una radiación solar intensa, dichos contaminantes son el resultado del proceso de combustión que se efectúa tanto en las industrias como en el mismo tráfico.

Es de suma importancia no confundir al ozono troposférico con el estratosférico, ya que este último actúa de forma beneficiosa absorbiendo la dañina radiación ultravioleta proveniente del sol, evitando así que traspase a la superficie de la Tierra. Estamos hablando del tipo de ozono causante de 17 mil muertes prematuras en Europa al año, 1700 de ellas en España. En el trascurso del 2017, el 30% de la población europea estuvo expuesta a niveles de O3 que excedieron el valor objetivo de la UE, según el último informe de la EEA. Todo esto sin mencionar las repercusiones que tiene en la vegetación, afectando la productividad de cultivos y bosques, alterando el crecimiento y el metabolismo de las plantas, acrecentando la sensibilidad de los arboles a los cambios de temperatura y ocasionando perdidas de producción en la agricultura en aproximadamente 5 a10% en toda la Comunidad Europea.

Además, es un gas de efecto invernadero, que contribuye al calentamiento global.

Es inconcebible que en los últimos 150 años hayamos modificado la estructura natural de nuestra atmósfera e hidrósfera más que en nuestros 200 mil años de historia. Está claro que se precisan cambios drásticos y normas muy estrictas si se quiere preservar la calidad de vida en el planeta. A causa de esta problemática, se reúnen 8 grandes organizaciones de la sociedad civil, impulsadas por el programa Horizonte 2020 y crean en conjunto el proyecto CAPTOR, un proyecto que consiste en el desarrollo de un nodo capaz de monitorear los niveles de ozono troposférico con redes de sensores de bajo coste. Las organizaciones encargadas del desarrollo del proyecto son las siguientes:

1) Grupo de investigación Statistical Analysis of Networks and Systems (SANS) de la Universitat Politècnica de Catalunya
2) Agencia Estatal Consejo Superior de Investigaciones Científicas (CSIC)
3) Ecologistas en Acción
4) Legambiente
5) Zentrum für Soziale Innovation (ZSI)
6) Global-2000
7) Université Blaise Pascal Clermont-Ferrand
8) Fundació privada per a la Xarxa Oberta, Lliure i Neutral Guifi.net

El proyecto CAPTOR combina los conceptos de ciencia ciudadana[1], aprendizaje colaborativo y activismo ambiental [2]para conseguir promover la

[1]La ciencia ciudadana incluye al público común en actividades científicas y promueve la participación de estos en la investigación a través de su conocimiento, herramientas y recursos.
[2]Activismo ambiental es la ideología de personas que colaboran a favor del medioambiente.

participación activa de los ciudadanos europeos para desarrollar el sentido de la responsabilidad, concienciar a la población europea acerca de la situación actual de la contaminación ambiental y lograr estimular el debate, propulsar procesos de reflexión y aprendizaje para buscar soluciones al problema de ozono, dirigirse a las autoridades con datos científicos fiables y sólidos de las estaciones de monitoreo y para transformar esta disputa en soluciones. CAPTOR dirige tres grandes programas piloto en Austria, Italia y España, tres regiones sumamente afectadas por la contaminación por ozono, impulsados por activistas de base y comunidades locales donde los ciudadanos participarán en el proyecto en diferentes niveles para abordar sus preocupaciones. Las tres regiones están ubicadas en:

- Barcelonès-Vallès Oriental-Osona (Cataluña, España)
- Pianura Padana (Valle del Po, Italia)
- Burgenland, Steiermark and Niederösterreich (Austria)

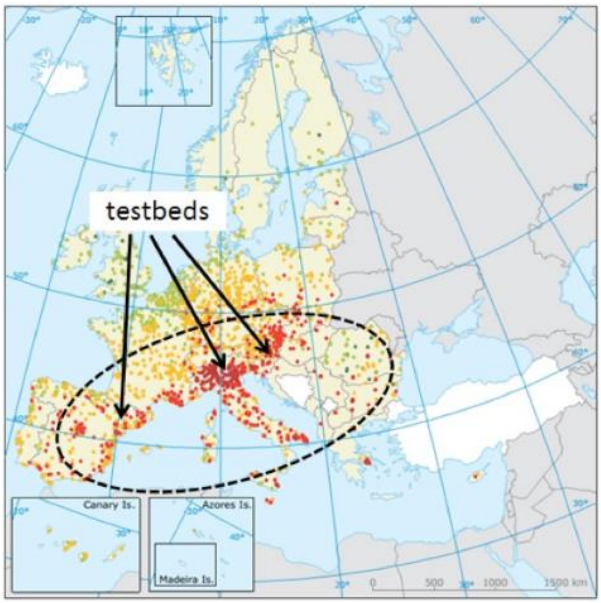


Figura 1. Regiones donde CAPTOR tiene cobertura.

Las campañas de desarrollo sustentable, como CAPTOR, son fundamentales para que adultos y jóvenes puedan tomar conciencia de la situación actual de la calidad del aire. Con la finalidad de que el proyecto siga operando el mayor tiempo posible, es necesario realizar un cambio del procesador interno de los nodos, ya que el Arduino Yun que incorporan actualmente, se ha descontinuado del mercado. Lo que se busca es un procesador potente, confiable, y los más importante, que vaya a perdurar muchos años más en el mercado.

1.2 Objetivos

1.2.1 Objetivo General

El objetivo principal de este proyecto no es otro que continuar con el desarrollo y mejoramiento de los nodos CAPTOR. Colaborando con el grupo de investigación SANS se le va a realizar una actualización hardware al modelo actual de los CAPTOR, sustituyendo la placa Arduino Yun, que traen incorporada todos los nodos, por la versión más reciente de Raspberry Pi. Para así, desarrollar un nodo más robusto, confiable y, sobre todo, perdurable.

1.2.2 Objetivos Específicos

Los objetivos específicos para lograr el objetivo general son los siguientes:

- Familiarizarse con el funcionamiento de los distintos periféricos que utiliza el nodo.
- Lograr adaptar y migrar todas las funcionalidades del CAPTOR a este nuevo prototipo con Raspberry Pi.
- Conseguir que la recolección de datos se efectúe con el menor porcentaje de error posible.

- Determinar y evaluar la eficacia, precisión, solidez y el potencial de la Raspberry.
- Desarrollar y preparar un nodo con las suficientes capacidades y potencia de cálculo para posibles mejoras futuras.
- Examinar y valorar que partes del código que se pueden aprovechar en esta actualización.
- Lograr que este nuevo prototipo CAPTOR funcione bajo un único código fuente de Python y no bajo una combinación de Python junto con Sketch de Arduino.

1.3 Estructura de la memoria

En seguida, se va a mostrar un sumario de cada una de las secciones en las que se va a dividir la memoria, con la finalidad de facilitar su entendimiento.

INTRODUCCIÓN. Se realiza una descripción del contexto en el que se va a desarrollar el trabajo, se define el problema, para posteriormente ubicarlo dentro de una problemática general y, se exponen las razones del porqué de la investigación. Así mismo, se mencionan los objetivos generales y específicos que se pretenden alcanzar.

ESTADO DEL ARTE. En esta sección se dan a conocer los trabajos y publicaciones previos relacionados a la problemática que busca resolver este proyecto, ya sea en libros, trabajos de investigación, artículos de revistas, etc.

ENTORNO DE DESARROLLO. Si se quiere realizar un buen prototipo CAPTOR, es necesario conocer a detalle el funcionamiento de cada uno de los periféricos que se va a utilizar. En esta sección, se hace una ardua investigación de todas las herramientas, procesos, elementos y técnicas que se necesiten saber para posteriormente implementarlas.

DESARROLLO DEL PROYECTO.

Se hace uso de todos los conocimientos adquiridos hasta el momento, para la integración de los periféricos, primero por separado y después en conjunto. En esta sección se explican a detalle todos los pasos seguidos, partiendo de configuraciones básicas hasta llegar a la integración de todo el código Python.

que las funcionalidades finales y el comportamiento general de los mismos, sea el deseado.

COSTES. Se realiza un recuento de todos los costes que involucran al proyecto, dichos costes se dividen en personal, hardware y costes indirectos.

CONCLUSION Y TRABAJO FUTURO. Se expone un conjunto de ideas sintetizadas tras la realización del proyecto. También se señala la posible implementación de otro elemento en un futuro cercano.

BIBLIOGRAFÍA. Enumeración de todas las fuentes bibliográficas consultadas para la elaboración del proyecto.

2
Estado del arte

En este capítulo se realiza una compilación acerca de otras investigaciones o proyectos que tengan objetivos y funcionalidades similares al proyecto en cuestión, en este caso analizaremos proyectos que desarrollen dispositivos que midan y registren valores de contaminación ambiental.

Durante las últimas décadas, la contaminación ambiental ha sido una gran fuente de preocupación para toda la humanidad. Existen distintas organizaciones interesadas en difundir toda información posible sobre este fenómeno. A continuación, se muestran algunos proyectos con funcionamiento similar al CAPTOR.

2.1 Proyecto Hiri

Es un sistema desarrollado por la Facultad de Ingeniería de la Universidad del Desarrollo (UDD) que ha dado bastante de que hablar en Chile y en varios países de Latinoamérica, ya que permite analizar la calidad del aire, reportarlo y en base a los resultados, tomar decisiones estratégicas. Lo denominaron Hiri, el Waze que mide la contaminación ambiental, un dispositivo constituido por pequeños sensores, de bajo costo y de fácil movilidad.

A diferencia del CAPTOR, este dispositivo puede ser utilizado tanto de manera móvil como fija y su funcionamiento consiste en que distintos usuarios vayan haciendo muestreos mientras se transportan por la ciudad o simplemente medir la exposición individual de la contaminación en distintos puntos de la ciudad, generando mapas tridimensionales y dinámicos de la calidad del aire; Con la finalidad que cualquier autoridad ambiental tenga acceso a los datos y puedan idear posibles soluciones.

Figura 2. Proyecto Hiri.

2.2 Proyecto piloto UPCT

Cuatro estudiantes de la Universidad Politécnica de Cartagena (UPCT) desarrollaron un aparato, alimentado por energía solar, que sube en tiempo real el porcentaje de contaminación de distintas avenidas y calles en Cartagena. Se utilizaron redes de IOT [3]como SigFox y LoRa[4], con el propósito de tener un consumo energético bajo y de poder enviar la información recaudada a grandes distancias.

Este proyecto está pensado para que ciclistas, conductores, corredores y peatones comunes, puedan conocer la calidad del aire por medio de una aplicación móvil y así puedan optar por tomar rutas no tan sobrecargadas de contaminación.

[3]IOT es el acrónimo de Internet of Things
[4]Lora y Sigfox son tecnologías de comunicación inalámbrica de largo alcance, baja potencia y baja velocidad.

Figura 3. Primer prototipo.

2.3 Iniciativa CIVITAS

El Observatorio del Medio Ambiente Urbano (OMAU) puso en marcha un proyecto piloto que permite registrar la contaminación de la ciudad, gracias a la colocación de dispositivos sensores en los autobuses municipales. Esta iniciativa nace con el propósito de aprovechar los recorridos que realizan los autobuses de la Empresa malagueña de transportes (EMT), para así tomar mediciones a lo largo de la ciudad y los usuarios puedan acceder a un mapa detallado con los puntos exactos donde se retienen grandes niveles de contaminación. Para esto, los dispositivos se sitúan en el techo de los vehículos y programados para coger medidas de óxidos de nitrógeno, monóxido de carbono, y ozono, cada hora durante el trayecto, ubicando todos los puntos mediante GPS.

Figura 5. Diseño del proyecto.

3

Entorno de desarrollo

En este capítulo se explican a detalle las herramientas que se van a utilizar para realizar esta actualización del nodo CAPTOR. En la sección 3.1 se analiza y justifica la utilización del lenguaje de programación Python. En la sección 3.2 se describen las funcionalidades y especificaciones que tiene la Raspberry Pi. La sección 3.3 trata del circuito integrado MCP30008. En la sección 3.4 se habla acerca del Reloj en tiempo real (RTC). Y finalmente, en la sección 3.5 y 3.6 se detalla el funcionamiento de los 2 tipos de sensores utilizados en el proyecto.

3.1 Python

Python es un lenguaje de scripting que surgió a principios de los 90, gracias al ingeniero de Guido Van Rossum; se caracteriza por ser un lenguaje interpretado, multiplataforma, orientado a objetos y con una sintaxis sumamente clara, sencilla y concisa; listo para desarrollar cualquier tipo de programa, desde software para aplicaciones científicas, servidores de red o incluso, páginas web.

3.1.1 ¿Por qué usar Python?

En los últimos años, la utilización de este lenguaje ha ido creciendo constantemente y en la actualidad es uno de los lenguajes de programación más usados por las distintas ventajas que ofrece.

Una ventaja destacable es que es un lenguaje de programación orientado a objetos, se trata de un paradigma que pretende modelar todo en función a objetos y a clases, en vez de lidiar con direcciones y registros de memoria; esto facilita el manejo y las posibles modificaciones que pueda sufrir el código. En pocas palabras, es un lenguaje con el más alto nivel de abstracción de lenguaje máquina, que ofrece un uso de conceptos de herencia[5], polimorfismo[6], encapsulación[7] y mucho más.

La finalidad detrás de Python que, además, es un lenguaje de código abierto, es equiparar la sintaxis lo mayor posible a la del ser humano, logrando una curva de aprendizaje sumamente corta. Un ejemplo claro de su sencillez es su

[5] Herencia es proceso donde una clase adquiere las propiedades y comportamientos de otra.
[6] Polimorfismo es la propiedad que hace factible el hecho de enviar mensajes sintácticamente iguales a distintos tipos de objetos.
[7] Encapsulación es un mecanismo que consiste en organizar y almacenar los métodos y datos de una clase, de tal forma que solo aquellas funciones que se guarden en dicha clase puedan tener acceso a los mismos.

indentado, ya que los bloques de código se definen mediante indentaciones, ahorrándose símbolos como llaves o palabras reservadas como BEGIN Y END, para obtener un código más corto y, sobre todo, legible. Otro ejemplo de suma importancia es su tipado dinámico, debido a que no es esencial declarar el tipo de dato que contiene una variable, este tipo de dato será definido en tiempo de ejecución según el valor que se le haya asignado a dicha variable.

Por otra parte, se debe recalcar que Python es un lenguaje interpretado con algunas partes compiladas, esto significa que se ejecuta usando un programa intermediario llamado interprete, donde se compila el código a un lenguaje comprensible para la máquina, paso a paso, instrucción por instrucción. La principal ventaja de esto es que permiten crear aplicaciones flexibles y más portables. El intérprete de Python está disponible en una gran cantidad de plataformas (Linux, Windows, UNIX, DOS, Solaris OS/2, Mac OS, etc.) por lo que si no se emplean las librerías determinadas que tiene cada plataforma, nuestro código será capaz de correr en todos estos sistemas sin modificaciones mayores.

Y finalmente, hay que mencionar la gran cantidad de funciones que vienen incorporadas para el manejo de archivos, strings, números, etc. Y a la gran comunidad que este lenguaje tiene detrás, encargándose continuamente de enriquecerlo, a través la creación de nuevas funciones y librerías, que permiten extender la funcionalidad de nuestras aplicaciones

3.1.2 Instalar Python

El trabajar con Python en Raspberry Pi no presenta mayor problema, puesto que la aplicación viene instalada por defecto con el sistema operativo Raspbian, del cual se hablará más adelante, por lo que no será necesario realizar ninguna instalación.

3.2 Raspberry Pi

Raspberry Pi es computador de placa reducida de bajo coste, desarrollado por la Fundación Raspberry Pi en Reino Unido, con el propósito de fomentar la enseñanza de informática y ciencias de comunicación. Este pequeño ordenador integra funcionalidades de electrónica como pines GPIO (Entrada/Salida de Propósito General), y de comunicación como UART (Transmisor-Receptor Asíncrono Universal), SPI (bus de interfaz de periféricos serie) y I²C (Circuito inter-integrado).

En la actualidad existen distintos modelos de Raspberry Pi en el mercado, entre ellos están el modelo A, B, B+, 2B, Zero, Zero W y el 3B+. Este proyecto se va a realizar con este último, puesto que presta una mayor cantidad de funcionalidades, como la inclusión de conectividad WiFi, que puede ser de utilidad en el futuro. A continuación, se muestra una imagen del modelo.

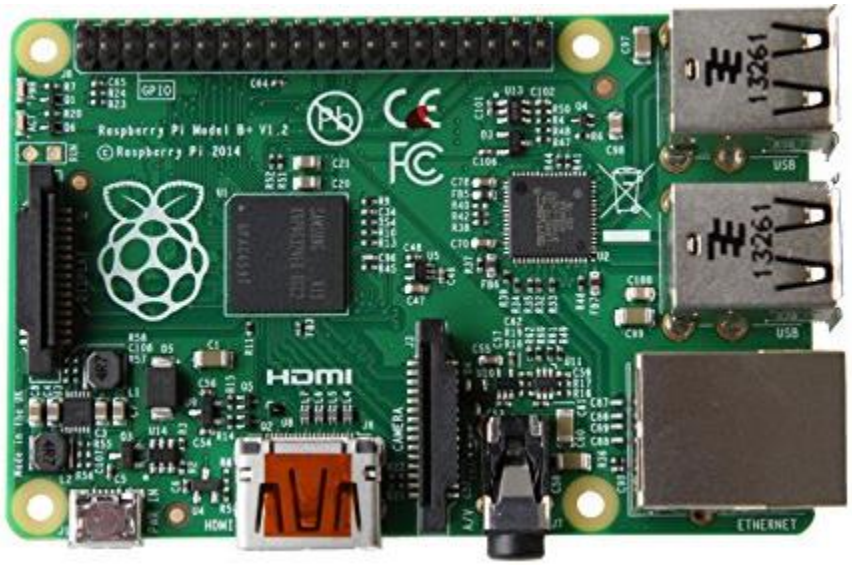


Figura 6. Raspberry Pi 3 Modelo B+.

La placa está diseñada para correr el sistema operativo GNU/Linux, que se caracteriza por ser un símbolo de la programación con código abierto. Esto

significa que el usuario es capaz de descargar el código fuente completo del sistema operativo y realizar los cambios que le convengan. Existen distintas distribuciones de Linux específicamente desarrolladas y optimizadas para el hardware de la Raspberry y en este caso se va a utilizar Raspbian, la distribución más conocida que existe y se basa en Debian, con todas las ventajas que ello conlleva. En pocas palabras, es una Debían pero que se encuentra optimizada para una arquitectura ARMv6 con soporte de coma flotante por hardware

3.2.1 Características técnicas

Como se mencionó anteriormente, el modelo utilizado en este primer prototipo será el 3B+, que posee las siguientes especificaciones.

Procesador	**Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz**
GPU	Broadcom VideoCore IV a 400 MHz con soporte de OpenGL ES 2.0
Memoria	1GB LPDDR2 SDRAM
Redes	Puerto Gigabit Ethernet, 2,4 GHz y 5 GHz 802.11b /g / n /ac WiFi
Bluetooth	Bluetooth 4.2, Bluetooth de baja energía (BLE)
Almacenamiento	Micro SD
GPIO	GPIO de 40 pines energía
Puertos	HDMI, toma de audio y video analógico de 3,5 mm, 4x USB 2.0, Ethernet, interfaz serie de cámara (SDI), interfaz serie de pantalla (DSI)

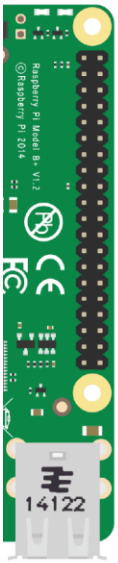
Dimensiones	82 mm x 56 mm x 19,5 mm, con peso de 50 gramos

Tabla 1. Características técnicas de Raspberry Pi 3 Modelo B+.

3.2.2 Pines GPIO

La Raspberry pi 3 modelo B+, además de comportarse como un ordenador, cuenta con 40 pines llamados GPIO (Entrada/Salida de Propósito General), que como su nombre lo indica, son pines que pueden configurarse como entradas o salidas digitales para controlar cualquier sensor, periférico o actuador externo, a través de distintos Scripts.

Los pines GPIO cuentan con una numeración específica, ya que no todos sirven para el mismo fin. En seguida, se muestra una imagen detallada de todos los puertos.



Peripherals	GPIO	Particle	Pin #			Pin #	Particle	GPIO	Peripherals
3.3V			1	X	X	2	5V		
I2C	GPIO2	SDA	3	X	X	4	5V		
	GPIO3	SCL	5	X	X	6	GND		
Digital I/O	GPIO4	D0	7	X	X	8	TX	GPIO14	UART
GND			9	X	X	10	RX	GPIO15	Serial 1
Digital I/O	GPIO17	D1	11	X	X	12	D9/A0	GPIO18	PWM 1
Digital I/O	GPIO27	D2	13	X	X	14	GND		
Digital I/O	GPIO22	D3	15	X	X	16	D10/A1	GPIO23	Digital I/O
3.3V			17	X	X	18	D11/A2	GPIO24	Digital I/O
SPI	GPIO10	MOSI	19	X	X	20	GND		
	GPIO9	MISO	21	X	X	22	D12/A3	GPIO25	Digital I/O
	GPIO11	SCK	23	X	X	24	CE0	GPIO8	SPI (chip enable)
GND			25	X	X	26	CE1	GPIO7	
DO NOT USE	ID_SD	DO NOT USE	27	X	X	28	DO NOT USE	ID_SC	DO NOT USE
Digital I/O	GPIO5	D4	29	X	X	30	GND		
Digital I/O	GPIO6	D5	31	X	X	32	D13/A4	GPIO12	Digital I/O
PWM 2	GPIO13	D6	33	X	X	34	GND		
PWM 2	GPIO19	D7	35	X	X	36	D14/A5	GPIO16	PWM 1
Digital I/O	GPIO26	D8	37	X	X	38	D15/A6	GPIO20	Digital I/O
GND			39	X	X	40	D16/A7	GPIO21	Digital I/O

Figura 7. Diagrama Pinout de Raspberry Pi 3 Modelo B+

Es importante mencionar que durante el transcurso del trabajo se va a estar trabajando con la numeración BCM[8]; es decir, los pines se van a numerar de acuerdo al pin GPIO correspondiente y no de acuerdo a la numeración que se basa en el orden de los pines de arriba a abajo de la placa.

Se puede observar que los pines GPIO 2 y 3 permiten configurarse como interfaz I^2C, una manera sumamente útil de comunicarse con distintos tipos de periféricos que funcionen bajo este protocolo. Posteriormente, se explicará cómo configurarlos.

Los pines GPIO 7, 8, 9, 10, 11 se pueden configurar como la primera interfaz SPI (SPI0) para interactuar con periféricos externo que siguen este protocolo. De igual manera, se hablará de como configurar estos pines más adelante.

Los pines GPIO 14 y 15 permiten configurarse como interfaz UART para un puerto serie convencional. En realidad, ésta es la configuración que tienen por defecto en Raspbian, ya que la UART se usa como consola. El BCM 14 TXD mostrará una consola por defecto, que con el cable Serial adecuado, hace posible controlar la Raspberry Pi. El BCM 15 RXD recibirá comandos por defecto y los traspasará a una consola, lo que proporciona el control de la Pi con un cable Serial.

La salida PWM de los pines GPIO 16 y 18 es particularmente útil, si se quiere trabajas con dispositivos complejos que requieran tiempos altamente precisos.

Los pines GPIO 0 y 1 no están disponibles. Están generalmente reservados para la incorporación adicional de una memoria serie en las placas de expansión acorde a la especificación HAT EEPROM[9]. Cabe mencionar que son los únicos

[8]BCM - Número de pin Broadcom, comúnmente llamado "GPIO"
[9] EEPROM abreviatura de memoria de sólo lectura programable y borrable eléctricamente, es un chip de memoria capaz de retener su contenido sin energía.

pines que se configuran como salidas durante el arranque, ya que todos demás pines son inicialmente configurados como entradas para evitar inconvenientes.

Cuando se vayan a usar los pines GPIO para interfaz con hardware de cualquier tipo hay que poner especial atención en no dañar la propia Raspberry Pi. Es de suma importancia comprobar que estamos trabajando con los niveles de tensión y corriente adecuados, ya que los pines están diseñados para su utilización con circuitos de 3.3V y, al contrario que un Arduino Yun, estos no cuentan con ninguna protección de circuitería; así que bajo ningún concepto se debe introducir un voltaje superior, ya que esto puede dañar los pines y dejarlos inservibles. Otro factor para tomar en cuenta es que los pines de alimentación de 3.3 y 5 V, permiten una corriente máxima de 50 mA, por lo que, si la aplicación que se quiere realizar requiere de una corriente mayor, es indispensable agregar una fuente de alimentación que sea capaz de suministrar ese valor de corriente para evitar daños en la placa.

3.3 Conversor analógico digital

El mundo de los computadores es en su mayoría digital, es decir, todo se encuentra codificado mediante un sistema binario de ceros y unos. Sin embargo, el mundo, el mundo real, es analógico. El procesado analógico de las señales adquiere una gran importancia cuando se va a trabajar con instrumentos de medición, sistemas electrónicos, sistemas de comunicaciones, equipos de control, entre otros. Para que la Raspberry Pi sea capaz de interactuar con dispositivos análogos, necesita de un conversor externo. El conversor analógico digital o ADC, como su nombre lo indica, es un dispositivo electrónico que permite convertir una señal analógica en una señal digital.

Los ADC poseen diversas características y parámetros, como la resolución, el número de canales, el suministro de voltaje, la interfaz, tipo de empaquetado y consumo de energía. El hecho de decantase por uno u otro va a depender de las necesidades específicas de cada aplicación.

3.3.1 Selección del ADC MCP3008

En este caso en particular se requiere de un conversor que tenga por lo menos 5 canales de entrada, ya que el nodo CAPTOR actual incorpora 5 sensores analógicos; pero que también quede con entradas libres para la posible implementación de más sensores en el futuro. También se dista de un conversor con una buena resolución para reducir el porcentaje de error a la hora de realizar una lectura. La resolución es el valor que rige cual es la mínima variación de voltaje que el conversor analógico digital puede medir a su entrada.

3.3.2 Características técnicas

El conversor MCP3008 es un chip de aproximaciones sucesivas y como se observa en la Tabla 2, cumple con las características antes mencionadas.

Voltaje de operación	**2.7 a 5.5 V**
Número de canales	8
Resolución	10 bits
Frecuencia de muestreo	200 kSPS
Temperatura de Trabajo	40 a 85 °C
Interfaz de Datos	Serie, SPI
Número de pines	16

Dimensiones	22x17 mm

Tabla 2. Características técnicas del ADC MCP30008.

3.3.3 Funcionamiento del MCP3008

Los ADC de aproximaciones sucesivas, como el MCP3008, son los más utilizados debido a su alta velocidad de conversión. El proceso de conversión que ejecutan este tipo de conversores se basa en la ejecución de comparaciones sucesivas de forma ascendente o descendente, hasta hallar una combinación que iguala el voltaje de la entrada con el del ADC.

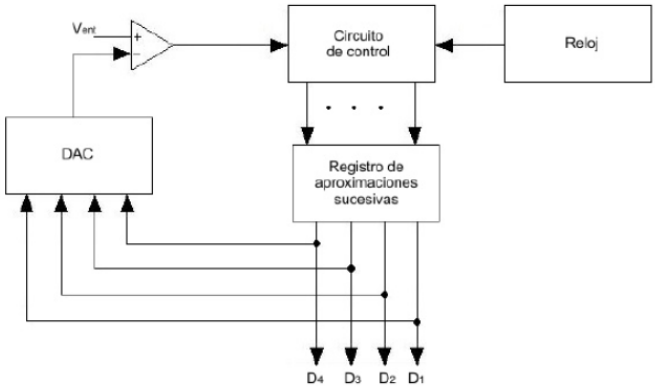


Figura 8. Diagrama de bloques.

De inicio, el registro de aproximaciones sucesivas le asigna el valor de 1 al bit más significativo y empieza con la comparación del voltaje de entrada. El código es introducido al DAC y este compara el voltaje de entrada con el valor de referencia entre dos, obteniendo una nueva señal, el valor de referencia suele ser de 3.3 o 5 V. Se pueden producir dos casos a partir de este procedimiento.

Si esta nueva señal es mayor al voltaje de entrada, el comparador hace que el registro de aproximaciones haga un reseteo al bit más significativo, es decir, sustituye el 1 que tenía asignado ese bit, por un 0 y le asigna al segundo bit más significativo el valor de 1.

Por otro lado, si la señal es menor al voltaje de entrada, el comparador realiza ningún cambie en el registro.

Posteriormente, se le asigna el valor de 1 al siguiente bit más significativo y se repite el mismo procedimiento, hasta que todos los bits hayas sido analizados. Al final de la conversión, se obtendrá un código binario, que principalmente, es una aproximación digital del voltaje de entrada muestreado.

3.3.4 Descripción de los pines

La figura 9, muestra la distribución de los pines del conversor MCP3008, los cuales se describen a continuación:

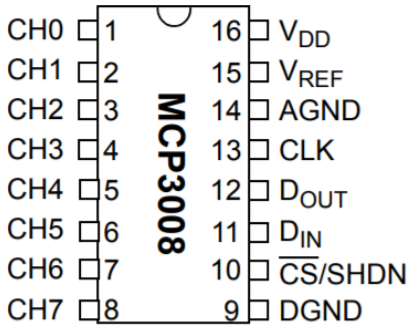


Figura 9. Asignación de pines del MCP3008.

Como se puede observar de lado izquierdo del circuito integrado, se tienen los 8 canales de entradas analógicas. En el lado derecho se encuentran el voltaje de alimentación VDD, que en esta ocasión será de 3.3 V y el VREF que es el voltaje

de referencia analógica, utilizada para cambiar la escala, como en este caso se quiere la escala completa, lo conectaremos de igual forma a 3.3 V. También se hallan los pines de CLK, DOUT, DIN y CS/SHDN, que se utilizaran para establecer la comunicación mediante el protocolo SPI, del que se hablará en la siguiente sección.

3.3.5 Protocolo de comunicación SPI

El protocolo de comunicación serial SPI es un bus síncrono que consta de 3 líneas, dos de ellas se encargan de transferir los datos y otra es el reloj, lo que permite que los dispositivos conectados trabajen en completa sincronización. Cada una de las líneas conduce la información entre los dispositivos que estén conectados al bus y cada dispositivo apareado al bus es capaz de tomar el lugar de receptor y trasmisor al mismo tiempo, a este tipo de comunicación serial se le denomina full duplex.

El proceso de intercambio de información se ejecuta en el momento preciso gracias a la presencia del reloj, ya que este le indica al receptor cuando tiene que adquirir el bit de la línea de datos que envía el dispositivo transmisor. Cuando se recibe el pulso, el receptor en seguida coge el dato de la línea y lo almacena en un registro de corrimiento. Una de las causas por las que el protocolo SPI es tan utilizado es porque el hardware de recepción que lo compone es este sencillo registro de corrimiento, el cual resulta ser una solución mucho más sencilla y asequible que los chips UART[10] de comunicación serie.

[10]UART es la abreviatura de Transmisor-Receptor Asíncrono Universal, es un dispositivo programable que controla los puertos y dispositivos serie.

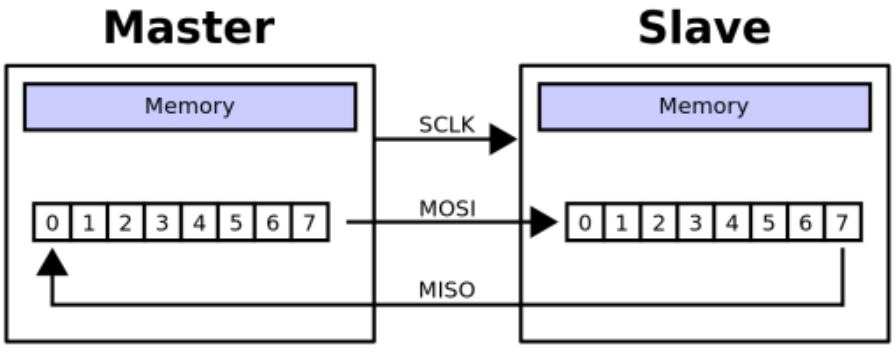


Figura 10. Transferencia de datos del protocolo SPI.

Dentro de este protocolo se determina un maestro cuya función será la de transferir toda la información hacia los dispositivos esclavos y estos únicamente se encargarán de recibir los datos y de ser controlados por su maestro. Los distintos esclavos se habilitan mediante una línea selectora denominada Chip Select, generalmente cada esclavo tiene su línea de selección individual.

En la Figura 10 se pueden ver las 3 líneas lógicas encargadas de efectuar todo el proceso. Es importante mencionar que las líneas del bus transmiten en una única dirección.

1) MOSI (Master Out Salve In) es la línea empleada para llevar los bits del maestro hacia el esclavo.
2) MISO (Master In Slave Out) es la línea encargada de llevar los bits del esclavo hacia el maestro.
3) CLK es la línea por la que se manda la señal de reloj que permite sincronizar los dispositivos.

3.4 Reloj en tiempo real

Un reloj en tiempo real (RTC) es un dispositivo electrónico de bajo consumo, capaz de mantener la fecha y hora actuales y la guarda para su posterior

consulta o manipulación, sin importar que este haya sido desconectado de la alimentación. Con la aparición del Internet de las cosas y los dispositivos usables, es indispensable, que los nuevos pedidos de instrucción sean más compactos, más económicos y, por sobre todas las cosas usar menos energía.

El término "Reloj en Tiempo Real" se creó para poder distinguir a este tipo de relojes de los relojes electrónicos convencionales, los cuales únicamente efectúan un conteo de pulsos de una señal, sin existir una relación directa con unidades temporales. La ventaja de usar estos circuitos integrados radica en su bajo consumo de energía y en la posibilidad de liberar una cierta carga de trabajo al procesador del sistema para que pueda dedicarse a tareas más críticas.

3.4.1 Selección de módulo DS1307

En el mundo de la electrónica existe una gran cantidad de módulos RTC. Entre los grandes fabricantes se destacan Texas Instruments, Maxim Integrated y Phillips. Y en cuanto a los modelos de relojes de tiempo real más utilizados para este tipo de aplicaciones, en las que se precisa una comunicación con microcontroladores como Raspberry Pi o Arduino Yun, destaca el módulo DS1307, siendo el que más se adecúa a las necesidades del proyecto. El DS1307 de Maxim Integrated, es una elección extremadamente útil cuando se necesita trabajar con eventos que demanden puntualidad y exactitud a lo largo del tiempo.

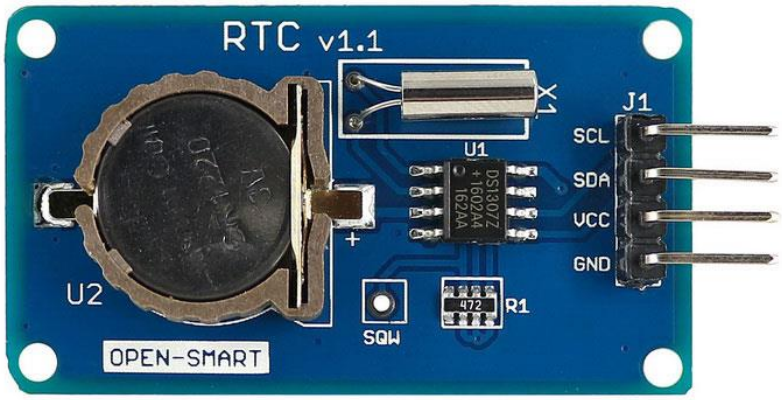


Figura 11. Chip DS1307.

El procedimiento que realiza la Raspberry Pi tras un reinicio es que se conecta a un servidor llamado NTP para consultar y actualizar la hora del sistema; pero el problema radica cuando esta no puede conectarse al servidor debido a la falta conectividad a internet. En este tipo de aplicaciones, en donde la placa va a sufrir varios reinicios a lo largo de los días, es primordial el uso de este módulo RTC, para que sea capaz de conservar la fecha y hora actualizadas en todo momento.

3.4.2 Características generales

El DS1307 es un potente reloj calendario en BCD, en seguida, se muestran las características generales extraídas de su datasheet.

- El reloj cuenta los segundos, minutos, horas, fecha del mes, mes, día de la semana, y año. Ajustando de forma automática los meses de 31 días y años bisiestos y es válido hasta 2100.
- Funcionamiento con ambos formatos de hora, ya sea 24 o 12 horas con indicador AM/PM.
- 56-Byte, con respaldo de batería, no volátil de RAM para almacenamiento de datos

- Interface Serie I^2C.
- Rango de voltaje de 3.3 V a 5 V.
- Onda-Cuadrada programable de la señal de salida.
- Detector Automático Fallo-Energía y Circuito Conmutación.
- Consume menos de 500nA en modo respaldo, modo de copia de seguridad con el oscilador funcionando.
- Rango de temperatura Industrial Opcional: -40 ° C a +85 ° C.
- Disponible en paquete de 8-pines DIP.

3.4.3 Funcionamiento del DS1307

El módulo DS1307 utiliza un cristal externo que genera una frecuencia de 32.768 KHz, este valor proviene de 2^15 = 32,768. Esto significa que el valor de frecuencia es divisible binariamente para generar un segundo exacto. El cristal ya viene incluido en el RTC de fábrica.

Este RTC necesita dos fuentes de alimentación. Por un lado, requiere de una fuente de alimentación de entre 3.3 a 5 V, que se consume mientras el circuito se encuentra encendido y funcionando. La otra fuente es la que procede de una batería de litio CR2032, conocidas como tipo reloj, que permitirá al reloj seguir funcionando cuando la alimentación principal haya sido desconectada. El cambio entre ambas fuentes de alimentación se ejecuta de forma automática.

Para la comunicación a través de bus I^2C, el DS1307 necesita de dos resistencias pull-up conectadas a los pines SCL y SDA. Si se requiere utilizar el pin SQW/OUT, este también necesita una resistencia pull-up para correcto funcionamiento. Este chip opera como un dispositivo esclavo en el bus serial I^2C.

3.4.4 Descripción de los pines

La figura 12, indica la distribución de los pines del módulo DS1307, los cuales se describen a continuación:

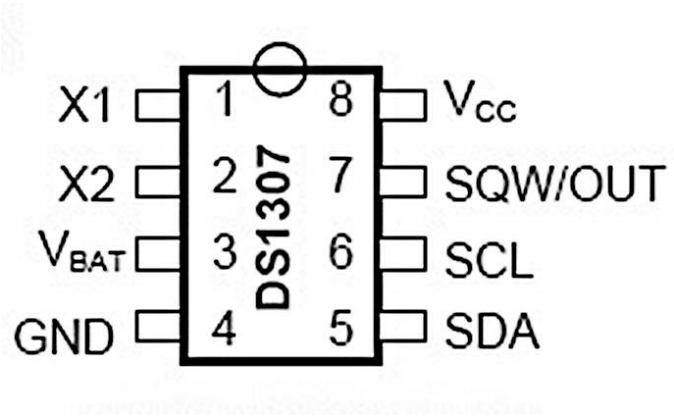


Figura 12. Asignación de pines del DS1307.

VCC, GND: La alimentación DC del dispositivo es proporcionada través de estos pines. VCC es la entrada de 3.3 a 5 V, mientras que GND es la referencia. Cuando se aplica un suministro de voltaje dentro del rango establecido, el chip será capaz de escribir y leer datos. Sin embargo, cuando se trabajen con voltajes inferiores a los 1.25 V, se deshabilitará la escritura y lectura de los datos. La función de la hora normal no se verá perjudicada por bajas tensiones de entrada.

VBAT: Entrada de alimentación de una pila de litio estándar de 3 Voltios. El voltaje se debe mantener entre los 2 y 3.5 V para un funcionamiento adecuado. Sí se introduce una batería de litio de 48mAh o más, mantendrá al RTC DS1307 funcionando por lo menos 10 años en ausencia de energía.

SCL: Es la entrada de reloj que sirve para sincronizar la transferencia de datos en la interfaz serie.

SDA: Es la línea de entrada/salida de los datos entre los dispositivos.

X1, X2: Conexiones para el cristal de cuarzo estándar de 32.768 KHz. El circuito interno está hecho para oscilar con un cristal con una capacitancia de carga especifica de 12.5 pF.

SQW/OUT: Al momento de activarse, el bit SQWE se fija en 1 y el pin SQW/OUT generar cuatro posibles frecuencias de salida: 1Hz, 4KHz, 8KHz o 32KHz. Este pin es de drenaje abierto, por lo que precisa de una resistencia pull-up externa.

3.4.5 Protocolo de comunicación I^2C

Es un protocolo creado para lograr una comunicación efectiva entre distintos circuitos que trabajas a bajas velocidades. El protocolo I^2C es uno de los más usado a la hora de comunicarse con sensores digitales, puesto que al contrario que el puerto Serial, su arquitectura posibilita adquirir una confirmación de los datos recibidos.

En la comunicación I^2C se utilizan únicamente dos hilos, a los que se le atribuye el nombre de bus I^2C. Por uno de los hilos SCL se mandará una señal de reloj para la sincronización, la señal representará la velocidad a la que trasfiere la información; y por el segundo hilo SDA se realiza el intercambio bidireccional de información. Se puede observar que este bus de comunicación presenta una ventaja frente al bus SPI, ya que solo necesita de dos cables para establecer la comunicación; pero, por el contrario, el funcionamiento se torna más complejo.

El bus I^2C posee una arquitectura de tipo maestro-esclavo. El maestro es el que comenzará la comunicación con los esclavos, pudiendo recibir o enviar datos a los esclavos. En cambio, los esclavos no son capaces de inicializar la comunicación, ni comunicarse entre sí directamente. Cabe mencionar, que los maestros no pueden hacer uso de un mismo puerto I^2C.

La metodología de comunicación de datos del bus I²C se realiza en serie y es sincrónica. Esto significa que el maestro suministrará una señal de reloj para que todos los dispositivos del bus se mantengan sincronizados. Gracias a esto, se hace innecesaria la inclusión de relojes para cada dispositivo y el hecho de tener que determinar una velocidad de transmisión y mecanismos específicos para que la transmisión se conserve sincronizada.

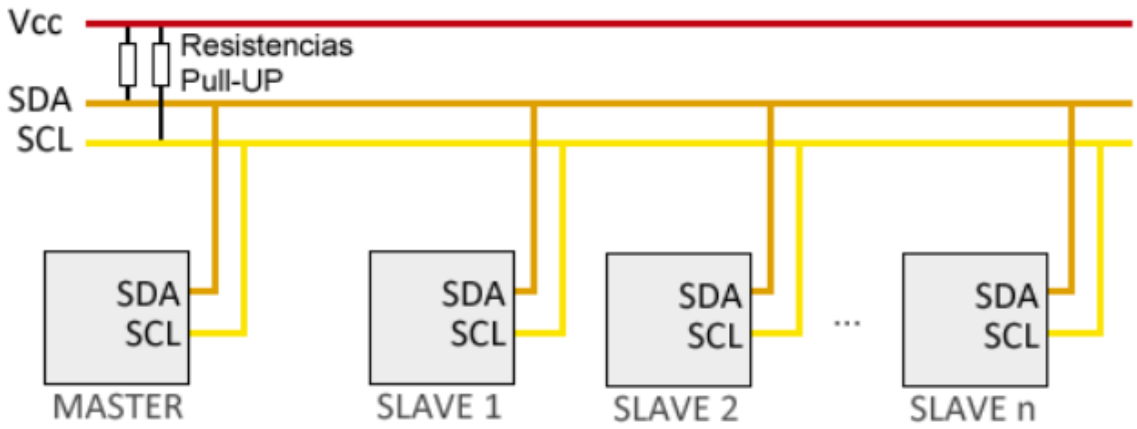


Figura 13. Conexionado para comunicación I²C.

En el bus I²C cada uno de los dispositivos tiene su dirección específica para poder acceder a ellos de forma individual. La asignación de esta dirección dependerá entera y completamente del usuario, ya sea que quiera fijarla por software o por hardware, modificando de los 3 últimos bits mediante jumpers. De esta forma, cuando el maestro requiera comunicarse con el esclavo, únicamente necesita enviar la dirección que se le haya establecido al esclavo mediante el bus I²C.

3.5 Sensor de Temperatura y Humedad DHT11

El DHT11 es un sensor de temperatura y humedad de bajo consumo y costo que se caracteriza por su alta estabilidad y fiabilidad a lo largo del tiempo. Están compuestos por dos sensores resistivos, uno capacitivo de humedad y un termistor NTC[11], para ser capaces de medir el aire circundante. A diferencia de otros sensores de temperatura como el LM35, este sensor utiliza un único pin digital calibrado para el intercambio de información y, por consiguiente, está más protegido delante del ruido y hace más sencilla la obtención de las mediciones desde un microprocesador como la Raspberry Pi.

Todos los sensores DHT11 se calibran rigurosamente en un laboratorio para adquirir la máxima precisión posible. Los coeficientes de calibración se guardan en forma de programas en la memoria OTP[12], para después utilizarse por el proceso interno de detección de señales.

3.5.1 Especificaciones técnicas

Las características técnicas del modelo DTH11 se muestran en la siguiente tabla.

Voltaje de operación	**3 a 5.5 V DC**
Rango de medición de temperatura:	0 a 50 °C
Precisión de medición de temperatura	±2.0 °C

[11]NTC es un elemento con un coeficiente de temperatura negativo respecto a la variación de su resistencia.

[12]OTP es una memoria no volátil de solo lectura, únicamente se puede programar una vez.

Resolución de la temperatura	1 °C
Rango de medición de la humedad	20% a 90% RH
Precisión de medición de la humedad	4% RH
Resolución de la humedad	1% RH
Periodo de sensado	2 segundos
Dimensiones	12 x15.5 mm

Tabla 3. Características técnicas del sensor DHT11.

Como se puede observar en la tabla anterior, una posible desventaja que pueden suponer este tipo de sensores es su velocidad de lectura y el tiempo que hay que esperar entre cada medición, ya que realizan una nueva lectura cada 2 segundos. Sin embargo, en este tipo de aplicaciones donde el sensor se programa para tomar medidas cada minuto, esa desventaja no representa ningún inconveniente.

3.5.2 Descripción de los pines

A continuación, se muestra una imagen con los distintos pines del sensor DHT11.

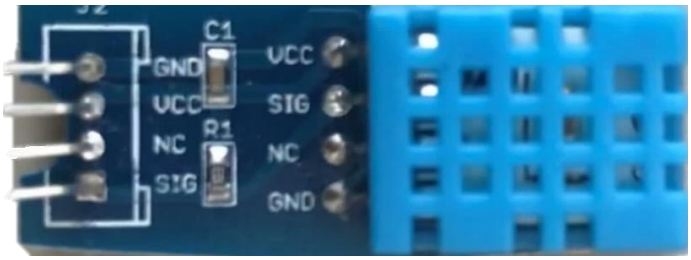


Figura 14. Sensor DHT11.

Se puede observar que el sensor únicamente cuenta con 4 pines, de los cuales uno se deja al aire. El pin VCC se conecta a la alimentación, el pin GND es la conexión a tierra y el pin GIG es el pin de trasmisión de datos. Cabe destacar que el modelo utilizado trae incorporado un PCB[13] que cuenta con una resistencia pull-up conectada entre el cable de datos y el de alimentación. La resistencia pull-up puede tener un valor de entre 4.7 K a 10 K Ohms.

3.6 Sensor de Ozono MICS-2614

El MICS-2614 es un sensor de pequeñas dimensiones compuesto por elementos pasivos y activos micro fabricados. En otras palabras, es un sensor MEMS[14] que consta de un diafragma micro mecanizado con una resistencia de calentamiento integrada y una capa de detección en la parte superior. El MICS es un dispositivo analógico que posee una estructura de metal-oxido MOS[15] y cuta aplicación es la detección del ozono O3. En cuanto a sus ventajas, se puede señalar su alta sensibilidad, amplio rango de detección, rápida respuesta térmica y una alta resistencia a los golpes y vibraciones.

3.6.1 Características técnicas

En la siguiente tabla se muestras las características técnicas principales de este sensor.

Voltaje de operación	**4.9 a 5.1 V**
Rango de detección	10-1000ppb
Temperatura ambiente de operación	-40 a 50 °C

[13]PCB es una placa de circuito impreso que conecta y soporta diversos componentes electrónicos.
[14]MEMS acrónimo de Sistemas Micro-Electro-Mecánicos.
[15]MOS acrónimo de un Semicondictor de Metal Oxido.

Detección de resistencia en el aire	3 a 60 KOhms
Factor de sensibilidad	1.5 a 4
Dimensiones	5 x 7 mm

Tabla 4. Características técnicas del MICS-2614.

3.6.2 Descripción de los pines

El diagrama de conexión de los pines se puede dividir en dos etapas como se aprecia en las siguientes figuras.

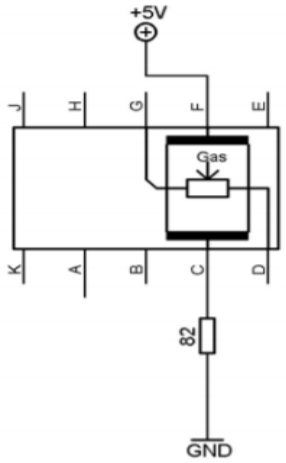


Figura 15. Circuito de alimentación.

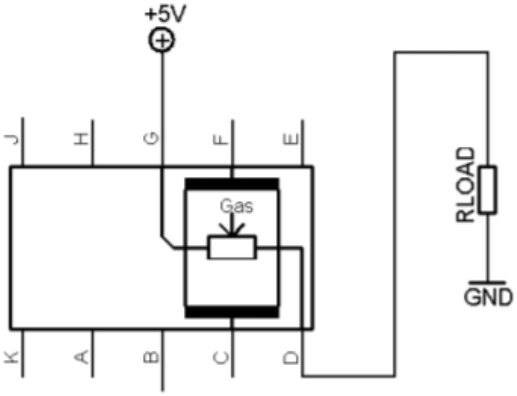


Figura 16. Circuito de medición.

La primera etapa es el circuito de alimentación, el pin F tiene que ir conectado a una fuente de 5 V y pin C va conectado a tierra, anexando una resistencia de 82 Ohm entre medio. La integración de esta resistencia es esencial para que el calentador interno pueda alcanzar la temperatura necesaria y funcionar correctamente.

La segunda etapa es el circuito de medición, el pin G de igual manera de conecta a 5 V y el pin D a tierra con una resistencia de carga de por medio. La resistencia sensible interna del sensor va a poder ser leída gracias a la incorporación de esa resistencia de carga, ya que el valor del voltaje medido en la resistencia de carga está directamente relacionado a la resistencia interna del sensor. El valor de dicha resistencia debe ser de por lo menos 820 Ohm para no dañar la capa sensible del sensor.

4

Desarrollo del proyecto

En esta sección se describe cada uno de los pasos que se siguieron para desarrollar el proyecto. En la sección 4.1 se muestra el proceso de instalación de Raspbian. En la sección 4.2 se detallan dos métodos de conversión analógica. En la sección 4.3 se indica cómo implementar el MICS-2614. En Sección 4.4 se enseña a poner en funcionamiento el sensor DHT11. En Sección 4.5 se dice cómo leer la hora del RTC. En Sección 4.6 se enseñar a prender un led. En Sección 4.7 se explica cómo configurar el modem Huawei. En Sección 4.8 se realiza la unificación de los dos códigos fuente y en la sección 4.9 se ejecuta el programa final y se analizan sus resultados.

4.1 Instalación del sistema operativo

El primer paso que se debe de realizar es la instalación del sistema operativo en la Raspberry Pi, ya que por defecto no lo trae instalado. La instalación es muy sencilla y para conseguirlo se necesitan los siguientes elementos.

- Fuente de Alimentación MicroUsb de 5 V y de preferencia con una salida de 2.4 A.
- Tarjeta MicroSD clase 10, se recomienda de 16 Gb en adelante.
- Monitor y cable HDMI
- Ratón y teclado
- Cable Ethernet RJ45

Una vez que se tengan todos los componentes necesarios, se debe ingresar a la página oficial de Raspberry Pi (https://www.raspberrypi.org/downloads/).

Existen dos maneras de instalar el sistema operativo. La primera consiste en descargar la imagen del sistema operativo deseado, en este caso Raspbian, y montar la imagen del disco con la ayuda de algún de terceros. El segundo método y el más sencillo, es descargando un software llamado NOOBS, acrónimo para New Out Of Box Software, que es un asistente creado por la empresa Raspberry. Solamente se necesitar descargar el archivo NOOBS, como se observa en la Figura 17; descomprimir el archivo y copiarlo a la tarjeta microSD. Cabe mencionar que ambos métodos requieren que la tarjeta SD esté formateada en formato FAT32.

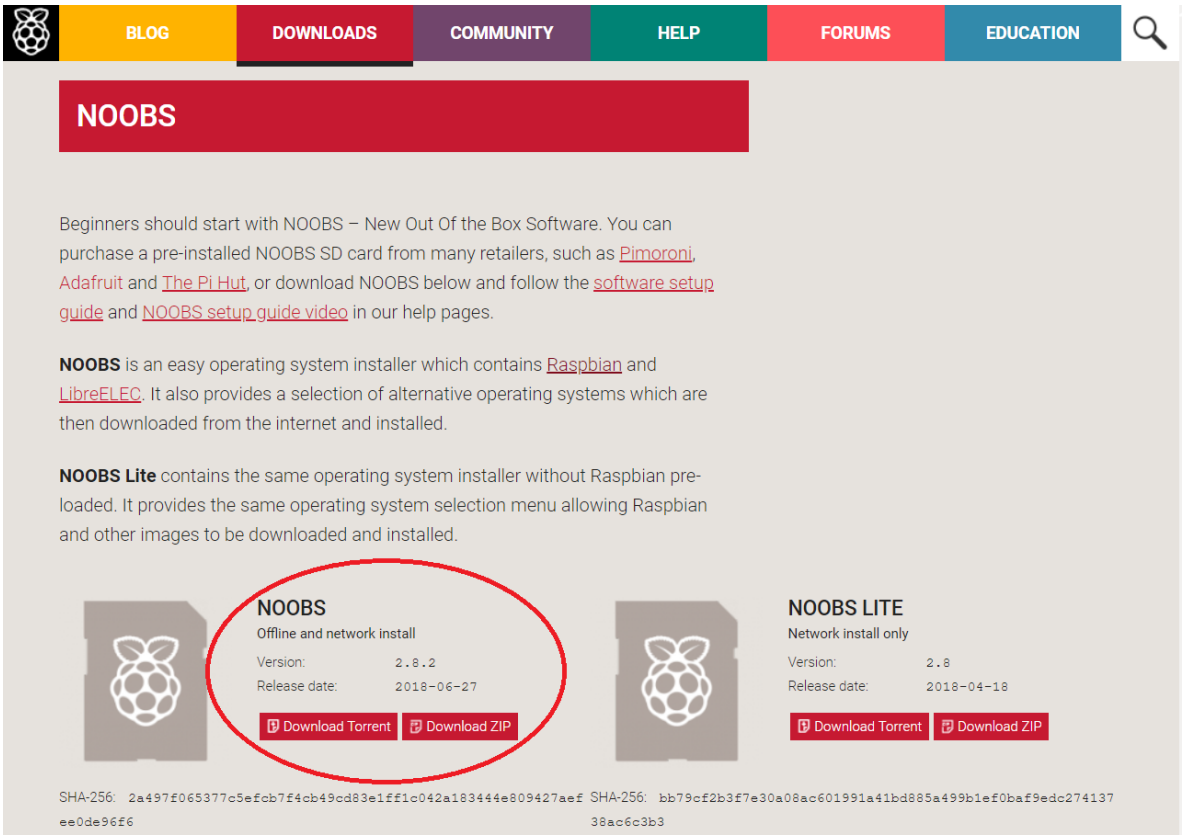


Figura 17. Descarga de NOOBS.

Se conectan los periféricos antes mencionados a la Raspberry junto con la microSD y se observa cómo se inicia el proceso de instalación, donde aparecerán una gran cantidad de sistemas operativos como Arch, RaspBMC, Raspbian, OpenELEC, entre otros. En esta ocasión se elegirá Raspbian y comenzará la instalación, este proceso puede tardar unos cuantos minutos.

Una vez finalizada la instalación, aparecerá una pantalla donde se pueden configurar algunas funciones opcionales de Raspbian, como, por ejemplo, configurar el teclado, cambiar la contraseña, modificar el idioma y la zona horaria, etc.

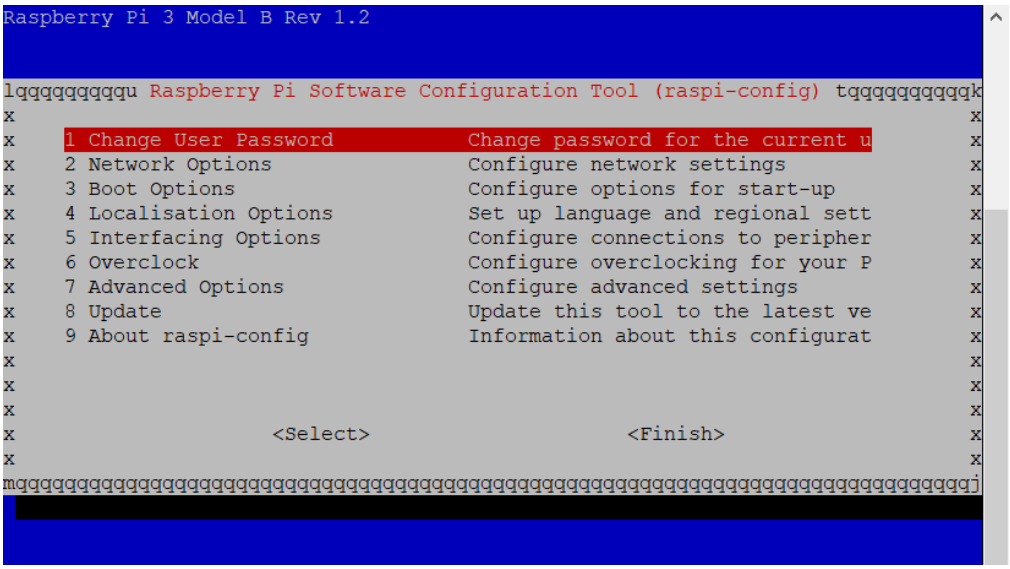


Figura 18. Menú de configuración de Raspbian.

Sin embargo, las modificaciones esenciales se encuentran en la pestaña de opciones de interface, que se observa en la Figura 18 y es donde se necesita habilitar las comunicaciones I²C y SPI, que por defecto vienen desactivadas. Cabe mencionar, que es posible acceder posteriormente a estas pantallas utilizando el comando *raspi-config*.

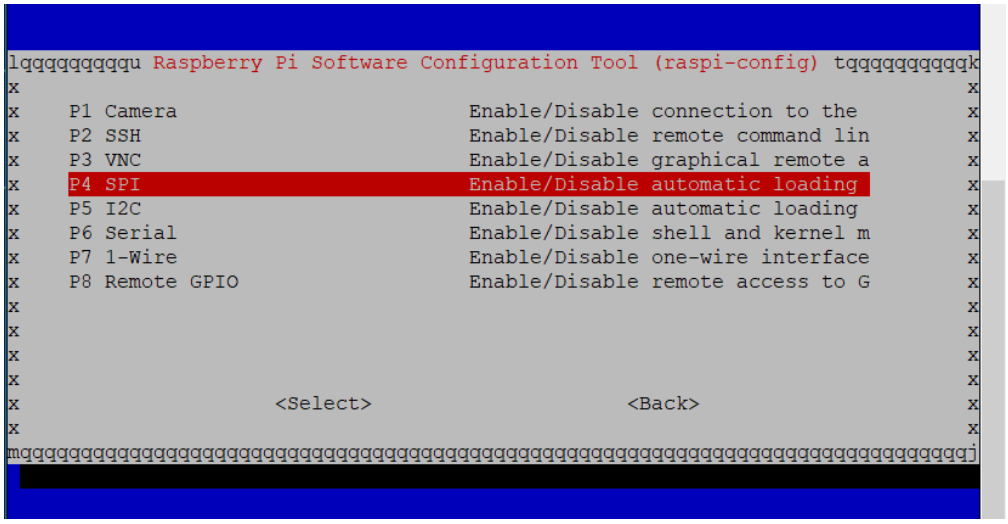


Figura 19. Opciones de interface de Raspbian.

En este momento, la Raspberry Pi ya está lista para utilizarse. Es esencial que, para futuras actualizaciones de software o descargas de paquetes y librerías, la Raspberry tenga acceso a internet, ya sea por WiFi o conectándose por cable a un Router u ordenador.

4.2 Lectura analógica digital

La Raspberry Pi tiene la capacidad de desarrollar una gran variedad funciones aplicaciones, como lo puede ser, el control de un motor de DC, la lectura digital de un botón pulsador, el parpadeo de un diodo LED. La única función que no es capaz de realizar es la leída de señales analógicas y esto representa un problema, pues los sensores principales que se van a utilizar en este proyecto son analógicos.

Sí, es correcto; la Raspberry no incluye ningún puerto analógico, ya sea de entrada o de salida, como sí ocurre con otras placas de la competencia, incluido la Arduino Yun que incorpora el CAPTOR actualmente. Sin embargo, existen dos métodos que permiten leer datos o señales analógicas. Uno es mediante software con la ayuda de un capacitor y el otro es a través de equipos externos, como lo puede ser el ADC MCP3008.

4.2.1 Técnica de respuesta escalonada

La técnica de respuesta escalonada consiste en analizar el comportamiento de un circuito RC al aplicarle un voltaje. La idea es aprovecharse de una propiedad electrónica básica de las resistencias y capacitores. Se toma un capacitor que inicialmente no tenga carga, y luego se conecta a la alimentación (como 3.3V) a través de una resistencia, el capacitor se irá cargando lentamente en un tiempo determinado. La clave reside en calcular el tiempo que tarda en

cargar el capacitor a través de un pin GPIO, es decir el tiempo que tarda en hacer la transición de bajo a alto.

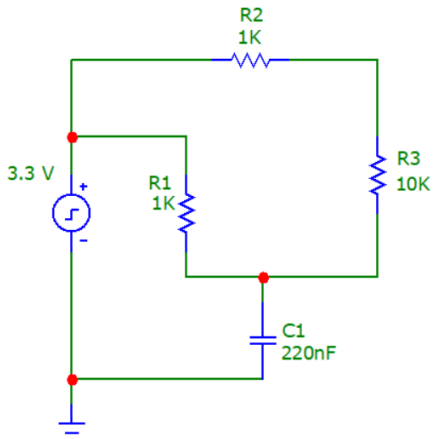


Figura 20. Circuito RC.

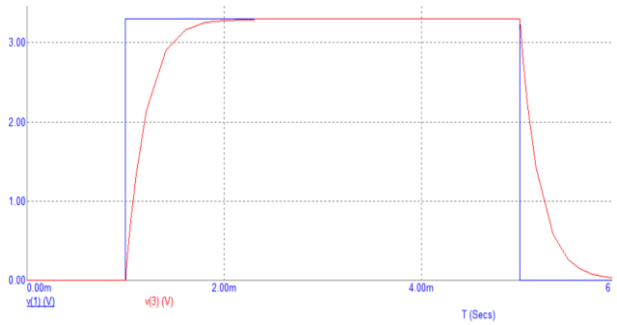


Figura 21. Señal de respuesta escalonada v(1) y carga-descarga v(3).

Como se puede observar en la Figura 21, la señal escalonada de entrada se aplica al circuito RC durante los primeros 5 milisegundos. Es decir, el capacitor C1 se carga durante ese tiempo a través de las resistencias R2 y R3 y cuando esté completamente cargado, le tomará un milisegundo en descargarse por completo mediante la resistencia R1. A medida que se produce este ciclo continuo de carga y descarga, se utiliza un contador de software en Python para obtener un valor equivalente a un dato analógico.

4.2.1.1 Montaje en la Raspberry

A continuación, se muestra a detalle cómo realizar el cableado correspondiente.

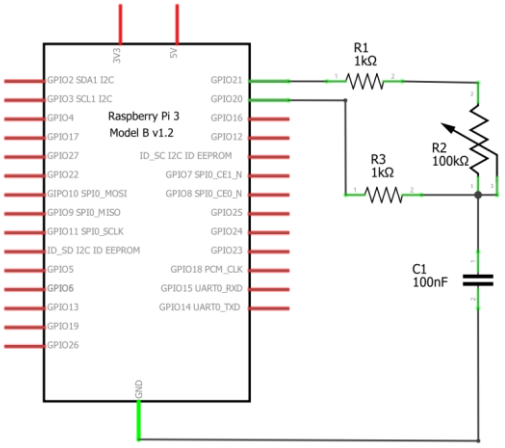


Figura 22. Diagrama esquemático del circuito.

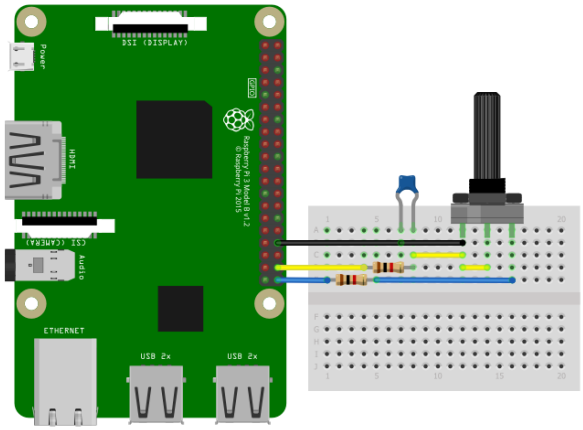


Figura 23. Circuito armado en Protoboard.

4.2.1.2 Funcionamiento y explicación del código Python

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
pin1 = 21
pin2 = 20
def discharge():
    GPIO.setup(pin1, GPIO.IN)
    GPIO.setup(pin2, GPIO.OUT)
    GPIO.output(pin2, False)
    time.sleep(0.005)
def charge_time():
    GPIO.setup(pin2, GPIO.IN)
    GPIO.setup(pin1, GPIO.OUT)
    count = 0
    GPIO.output(pin1, True)
    while not GPIO.input(pin2):
        count = count +1
    return count
def analog_read():
    discharge()
    return charge_time()
while True:
    print(analog_read())
    time.sleep(1)
```

Tabla 5. Código para técnica de respuesta escalonada.

Como se puede observar en la Tabla 2, lo primero es importar la librería que nos permite manejar y controlar los GPIO desde Python. Después se definen los pines GPIO que se van a utilizar; en este caso el GPIO 21 se utilizará para manejar la parte de carga del circuito y el GPIO 21 para la parte de descarga.

Se crea una función de descarga para poder leer los datos del capacitor, estableciendo el tiempo de 5 ms que se obtuvo mediante la simulación. Después se coloca una función de carga para registrar el valor del conteo analógico. Posteriormente, se crea una función de lectura analógica para leer los datos de carga y descarga. Y finalmente se imprime el valor en pantalla.

4.2.1.3 Análisis de resultados

```
pi@raspberrypi:~ $ python adc.py
11
11
9
7
5
6
6
10
10
10
7
5
9
9
8
11
11
11
7
6
7
9
11
```

Figura 24. Valores analógicos del potenciómetro.

Una vez que se obtuvieron las lecturas analógicas, se puede observar a primera instancia la fiabilidad de este método, ya que no proporciona mediciones estables. Cabe recalcar que el valor del potenciómetro siempre se mantuvo fijo.

Las posibles causas de la pésima exactitud de este método se pueden deber a que Python es un lenguaje interpretado, lo que significa que el tiempo de los bucles se ve afectado por el sistema operativo que realiza otras tareas en segundo plano. Esto seguramente está afectando al ciclo de conteo que se utilizó.

4.2.2 Método con ADC

En vista del rotundo fracaso del primer método, se optó por la incorporación del chip externo MCP3008 para que la Raspberry Pi sea capaz de leer entradas analógicas satisfactoriamente. Para corroborar que el ADC está funcionando correctamente se probará obteniendo el valor de una resistencia de 10 Kohms. En este paso se le dará uso al protocolo SPI que se habilitó en la pestaña de configuración de Raspbian.

4.2.2.1 Montaje en la Raspberry

En seguida, se enseña cómo debe de ser el cableado del ADC y de la resistencia.

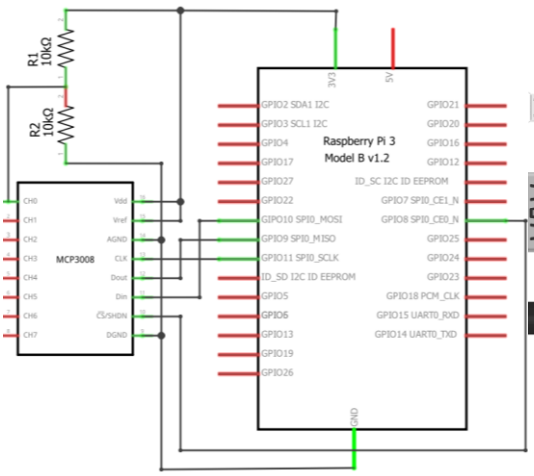


Figura 25. Diagrama esquemático del circuito.

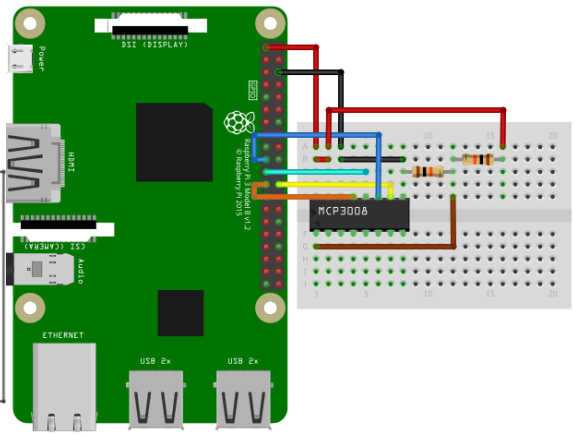


Figura 26. Circuito armado en Protoboard.

Antes de pasar a la siguiente sección, es importante verificar que no existan errores de cableado.

4.2.2.2 Funcionamiento y explicación del código Python

Como se comentó en las secciones pasadas, este ADC cuenta con 10 bits de resolución, lo que significa que tiene la capacidad de detectar 2^{10} o 1024 niveles analógicos discretos y estará trabajando a un voltaje de referencia de 3.3 V. En resumen, el convertidor asumirá que 3.3 V es 1023 y cualquier valor inferior a ese voltaje será una relación entre esos 3.3 V y 1023, se puede ver ejemplificado en la siguiente ecuación.

$$\frac{Resolución\ del\ ADC}{Voltaje\ de\ Referencia} = \frac{Lectura\ del\ ADC}{Voltaje\ analógico\ medido} \longrightarrow \frac{1023}{3.3\ V} = \frac{Lectura\ del\ ADC}{Voltaje\ analógico\ medido}$$

Programando esta ecuación en Python, se debería obtener un valor que se sitúa entre 0 y 1023; sin embargo, ese no es el valor que se está buscando. Para obtener el valor de la resistencia, en este caso la R2, se necesita hallar una expresión que permita relacionar el voltaje de salida V_{out} con el voltaje de entrada V_{in}. En otras palabras, se necesita aplicar un divisor de voltaje, que no es más que un circuito sencillo de resistencias en serie. La división que se hace va a ser proporcional a las resistencias involucradas en el divisor. La ecuación que define al divisor de voltaje y el despeje de la incógnita R2, son los siguientes.

$$Vout = Vin\frac{R2}{R1+R2} \longrightarrow R2 = R1\left(\frac{Vin}{Vout} - 1\right) \longrightarrow R2 = 10k\left(\frac{3.3}{Vout} - 1\right)$$

Donde V_{out} va a ser el valor analógico medido por el ADC.

Para que Python sea capaz de leer y manipular los datos que está mandando el conversor MCP3008, es obligatorio instalar los siguientes paquetes en la Terminal de la Raspberry.

```
sudo apt-get install python-dev python-pip
sudo pip install spidev
```

Para que se apliquen los cambios, se necesita reiniciar el dispositivo.

El paso siguiente será importar el módulo en el programa Python y activar el BUS para la comunicación SPI con el conversor analógico digital.

```
import spidev
import os
spi = spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz=1000000
```

Después se crea la función ReadChannel que recibirá el canal deseado y activará el bus SPI para poder recibir los datos del conversor. Los canales que se vayan a utilizar se deben de ser enteros del 0 al 7.

```
def ReadChannel(channel):
  adc = spi.xfer2([1,(8+channel)<<4,0])
  data = ((adc[1]&3) << 8) + adc[2]
  return data
```

Explicando un poco la lógica de la función, se tienen dos líneas notables.

La primera línea del spi.xfer envía 3 bytes al dispositivo. El primer byte tiene el valor de 1 que se representa 00000001 en binario. La parte de 8+channel sería un binario de 00001000, suponiendo que el canal sea 0 y el <<4 realiza un corrimiento de esos bits hacia la izquierda, obteniendo un resultado de 10000000. Después se manda un último byte con el valor de 0 o 00000000 en binario. Por lo tanto, se envía al dispositivo el valor de 00000001 10000000 00000000 y el dispositivo en respuesta, devuelve otros 3 bytes.

La línea "data =" extrae 10 bits de esa respuesta y esto representa la medición. La segunda línea de data se encargará de extraer 10 bits de la respuesta para así representar la medición final.

```
def ConvertVolts(data,places,r=10):
        volts = (data * 3.3) / float(1023)
        resistence = r * (3.3/volts - 1)resistence = round(resistence,places)
        return resistence
```

La función ConvertVolts se encarga de obtener el valor deseado de resistencia del sensor. Su funcionamiento se explicó anteriormente mediante las ecuaciones. El único anexo que se hizo es el redondeo del resultado final a dos decimales.

```
reschannel = 0
while True:
        resADC = ReadChannel(reschannel)
        resistencia = ConvertVolts(resADC,2)
        print("Resistencia: {}  \tOhms".format(resistencia))
        time.sleep(1)
```

Después en el ciclo infinito, se mandan a llamar las funciones previamente definidas, a la primera se le asigna el número de canal como parámetro y a la segunda el valor que leyó el ADC. Finalmente se imprime el resultado y se establece un delay de 1 segundo entre cada medición.

4.2.2.3 Análisis de resultados

```
pi@raspberrypi:~ $ python adc.py
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
Resistencia: 10.02 KOhms
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
Resistencia: 10.02 KOhms
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
Resistencia: 9.98 KOhms
```

Figura 27. Valores de la resistencia de 10 KOhms.

A diferencia del método de respuesta escalonada, en esta ocasión se puede notar que los valores de la resistencia apenas fluctúan, sin mencionar que se obtuvo un valor casi idéntico al esperado. No hay punto de comparación entre un método y el otro, de modo que ya se puede implementar este método para los sensores de ozono MICS-2614.

Se realizó la prueba primero con una resistencia para comprobar la veracidad y precisión del método, dado que, si se aplicaba directamente sobre el sensor de ozono, no se hubiera tenido la certeza de que el conversor analógico proporcionara unos valores correctos.

4.3 Implementación de sensores MICS-2614

La incorporación de los sensores MICS-2614 se coloca como una de las más importantes del proyecto, ya que, sin ellos no sería posible la lectura de los niveles de ozono del aire. Su implementación sigue el mismo principio de la resistencia en la sección pasada, así que se va a seguir la misma metodología.

4.3.1 Montaje en la Raspberry

A continuación, se exhibe como queda cableado de los 5 sensores MICS en la protoboard.

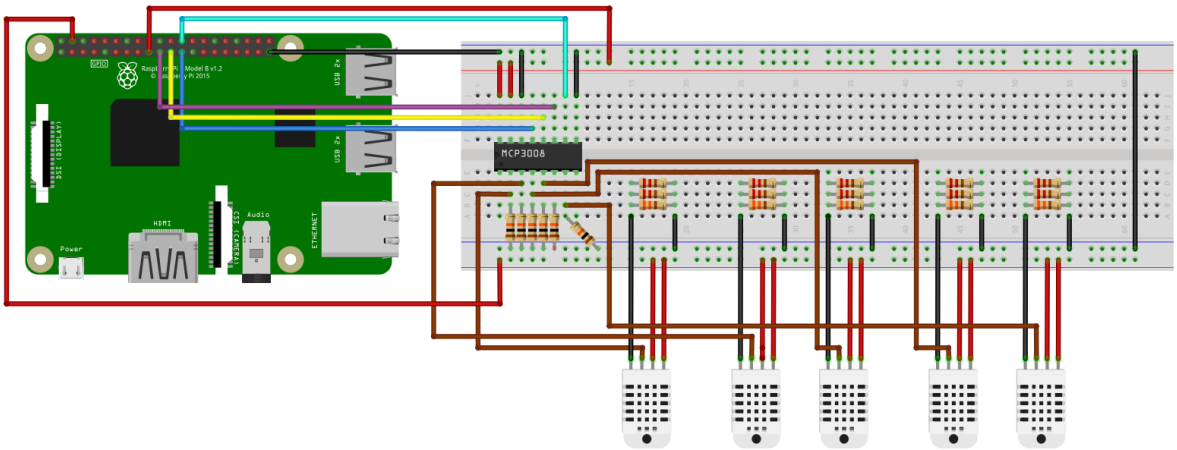


Figura 28. Circuito de sensores MICS-2614 armado en Protoboard.

Antes de pasar a la siguiente sección hay que comprobar que todo esté conectado correctamente. En este caso se manejan dos voltajes distintos pues los sensores necesitan una alimentación de 5 V.

Hay que recordar que para que sensor lea mediciones correctas es necesario que el calentador interno alcance una temperatura óptima, por ello es que se coloca un arreglo de 3 resistencias en paralelo, dos de 220 Ohms y una de 330 Ohms, para al final obtener un arreglo de 82.5 Ohms. También se le colocó una

resistencia de carga de 10 KOhms, que es a la que se le aplicará el divisor de voltaje que se mencionó anteriormente.

4.3.2 Funcionamiento y explicación del código Python

El código Python no sufre de grandes cambios aparentes, debido a que el fundamento es el mismo, así que no tiene sentido explicar nuevamente el funcionamiento. Simplemente al final del código se deben definir los 5 canales a utilizar, llamar a las funciones ReadChannel y ConvertVolts con dichos canales y finalmente imprimir los valores de resistencia de cada uno de los sensores, agregándole un delay de un segundo.

```
o31channel = 1
o32channel = 2
o33channel = 3
o34channel = 4
o35channel = 5
while True:
        o31ADC  = ReadChannel(o31channel)
        o31 = ConvertVolts(o31ADC,2)
        o32ADC  = ReadChannel(o32channel)
        o32 = ConvertVolts(o32ADC,2)
        o33ADC  = ReadChannel(o33channel)
        o33 = ConvertVolts(o33ADC,2)
        o34ADC  = ReadChannel(o34channel)
        o34 = ConvertVolts(o34ADC,2)
        o35ADC  = ReadChannel(o35channel)
        o35 = ConvertVolts(o35ADC,2)
        print("-----------------------------------------------------------------")
        print("O3,1 Sensor:  {} ({} KOhms)".format(o31volts,o31))
        print("O3,2 Sensor:  {} ({} KOhms)".format(o32volts,o32))
        print("O3,3 Sensor:  {} ({} KOhms)".format(o33volts,o33))
        print("O3,4 Sensor:  {} ({} KOhms)".format(o34volts,o34))
        time.sleep(1)
```

Tabla 6. Código para prueba del MICS-2614.

4.3.3 Análisis de resultados

```
-----------------------------------------
O3,1 Sensor:  328 (21.19 KOhms)
O3,2 Sensor:  491 (10.84 KOhms)
O3,3 Sensor:  670 (5.27 KOhms)
O3,4 Sensor:  718 (4.25 KOhms)
O3,5 Sensor:  663 (5.43 KOhms)
-----------------------------------------
O3,1 Sensor:  328 (21.19 KOhms)
O3,2 Sensor:  491 (10.84 KOhms)
O3,3 Sensor:  669 (5.29 KOhms)
O3,4 Sensor:  719 (4.23 KOhms)
O3,5 Sensor:  663 (5.43 KOhms)
-----------------------------------------
O3,1 Sensor:  327 (21.28 KOhms)
O3,2 Sensor:  490 (10.88 KOhms)
O3,3 Sensor:  669 (5.29 KOhms)
O3,4 Sensor:  718 (4.25 KOhms)
O3,5 Sensor:  662 (5.45 KOhms)
-----------------------------------------
O3,1 Sensor:  329 (21.09 KOhms)
O3,2 Sensor:  490 (10.88 KOhms)
O3,3 Sensor:  670 (5.27 KOhms)
O3,4 Sensor:  719 (4.23 KOhms)
O3,5 Sensor:  662 (5.45 KOhms)
```

Figura 29. Valores de resistencia de los MICS-2614.

Como se muestra en la Figura 29, ciertamente se obtienen 5 datos distintos que corresponden a cada sensor de ozono. Por el momento, son simplemente valores de resistencia o datos ROW y no existe forma de corroborar si son correctos o incorrectos, hasta aplicarles un valor de corrección.

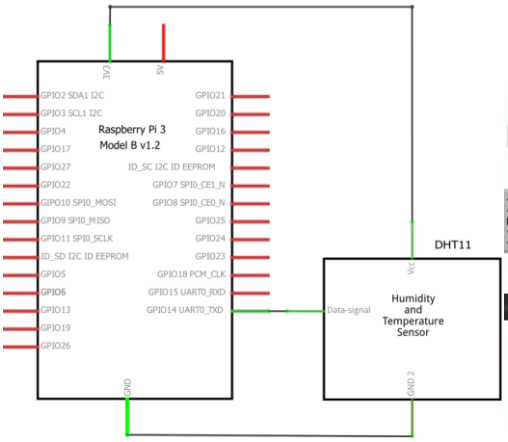
4.4 Implementación del DHT11

Continuando con la incorporación de sensores, se configurará el sensor DHT11 para poder medir la temperatura ambiente y humedad relativa. Un IC[16] [16]en la parte posterior del módulo es el encargado de convertir las medidas de resistencia de termistor y del sensor de humedad en sus respectivos valores digitales. La implementación es bastante simple, puesto que solo posee 3 pines funcionales.

[16] IC es una terminología de circuito integrado.

4.4.1 Montaje en la Raspberry

En seguida, se muestra el diagrama de esquemático y el circuito en protoboard.



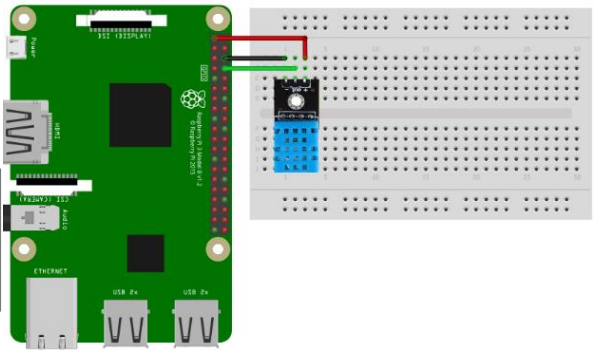


Figura 30. Diagrama esquemático del circuito.

Figura 31. Circuito armado en Protoboard.

Hay que recordar que este módulo con PCB ya trae incluida la resistencia pull-up de 10K entre VCC y el pin de datos, por eso ya no es necesario añadirla en el circuito.

4.4.2 Funcionamiento y explicación del código Python

El sensor DHT11 utiliza su sistema propio de comunicación bidireccional a través de un único pin, haciendo uso de señales temporizadas. Aunque se conecte a un pin digital, en el interior del sensor se efectúa la conversión de analógico a digital.

En cada lectura se envían un total de 40 bits que corresponden a los datos de humedad y temperatura, a una velocidad de 4ms.

0011 0101	0000 0000	0001 1000	0000 0000	0100 1001
8 bits humedad	**8 bits humedad**	**8 bits temperatura**	**8 bits temperatura**	**bits de paridad**

Figura 32. Información de temperatura y humedad representada en bits.

Como se puede contemplar en la Figura 32, la lectura se divide en tres grupos de bits.

La primera agrupación de bits corresponde a la parte entera de la humedad, mientras que la segunda representa la parte decimal. Lo mismo sucede con la segunda agrupación, solo que en este caso representa a la temperatura. Finalmente, la tercera agrupación se utiliza para hacer una comprobación de errores. Este grupo de bits se asegura de que la información se recibe correctamente, sumando los 4 bytes; esta suma tiene que ser igual a los bits de paridad. En este caso, la suma de 0011 0101 + 0000 0000 + 0001 1000 + 0000 0000 da como resultado 0100 1101, lo cual corresponde al valor de bits de paridad.

En resumen, para poder leer y manipular los datos del sensor en un programa, se necesita generar y recibir las señales temporizadas del protocolo DHT. Sin embargo, para simplificar el proceso se va a utilizar una librería que ya entrega el valor directo en decimal. La librería se descargará colocando estos comandos en la terminal.

```
sudo apt-get install git-core
git clone https://github.com/sebastianmartinezbasurto/dht11.git
sudo apt-get install build-essential python-dev
cd dht11
sudo python setup.py install
```

Con esto se habrá clonado el repositorio git de la librería dht11, se instalará el software Python-dev que permite agregar librerías Python a la Raspberry Pi, después se abre la carpeta que se creó al clonar con git y finalmente instala la librería.

```
import RPi.GPIO as GPIO
import dht11
GPIO.setmode(GPIO.BCM)
data = dht11.DHT11(pin=14)
while True:1
        result = data.read()
        print("Temperature: {} C ".format(result.temperature))
        print("Humidity: {} %".format(result.humidity))
        time.sleep(2)
GPIO.cleanup()
```

Tabla 7. Código para prueba del DHT11.

En cuanto al código Python, se empieza por importar la librería dht11 recién descargada y la que RPi.GPIO que posibilita el control de dichos pines. Seguidamente se inicializan los pines GPIO de la Raspberry, especificando con un paréntesis que se está trabajando con la numeración BCM y después se especifica el pin al que estará conectado el sensor, en este caso el GPIO 14.

En el while infinito, únicamente se llama a la función de lectura, que se va a encargar de realizar todo el proceso de conversión de binario a decimal. Posteriormente, se imprimen los resultados de temperatura y humedad. Finalmente se aplica una tasa de actualización entre lecturas de dos segundo, si este tiempo es menor, el sensor puede obtener lecturas erróneas.

Siempre que se trabaja con pines GPIO es importante limpiar los puertos mediante la función GPIO.cleanup(), lo que restablecerá los puertos que se hayan utilizado al modo de entrada y así evitar posibles daños en los pines.

4.4.3 Análisis de Resultados

```
Temperature: 28 C
Humidity: 33 %
Temperature: 28 C
Humidity: 33 %
Temperature: 28 C
Humidity: 33 %
Temperature: 28 C
Humidity: 33 %
Temperature: 28 C
Humidity: 33 %
Temperature: 28 C
Humidity: 33 %
Temperature: 28 C
Humidity: 33 %
Temperature: 28 C
Humidity: 33 %
Temperature: 28 C
Humidity: 33 %
Temperature: 28 C
Humidity: 33 %
Temperature: 28 C
Humidity: 33 %
Temperature: 28 C
Humidity: 33 %
```

Figura 33. Resultados de Temperatura y Humedad.

Analizando los resultados de la imagen 4, se comprueba que estos sensores digitales de temperatura y humedad son enormemente precisos y consistentes, tal y como lo dictan las especificaciones. Mientras no se exceda el intervalo mínimo de muestro, no se debería tener ningún problema con las mediciones.

4.5 Implementación del RTC DS1307

Hasta el momento, la Raspberry presenta un serio problema; si por algún motivo la placa se reinicia y esta no cuenta con conexión a internet, la fecha que mostrará el sistema al arrancar se remontará al año 1970. Evidentemente, para una aplicación donde se busca llevar un registro de la contaminación por un periodo de tiempo específico, es indispensable que la fecha siempre se mantenga actualizada. Con la finalidad de mantener la hora en todo momento, se incorpora el reloj en tiempo real DS1307.

4.5.1 Montaje en la Raspberry

A continuación, se presenta el diagrama esquemático de las conexiones.

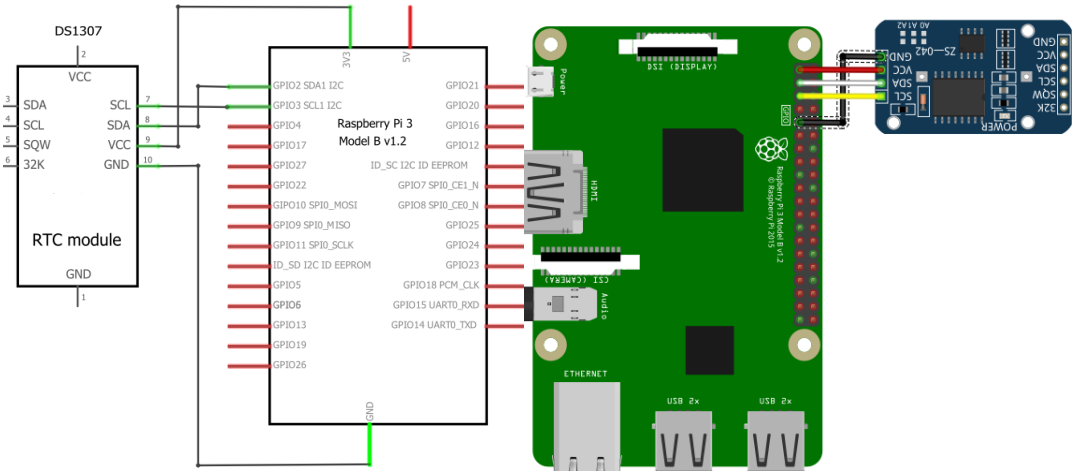


Figura 34. Diagrama esquemático del circuito.

Figura 35. Circuito armado en Protoboard.

Es necesario que verificar que la Raspberry Pi se encuentre apagada antes de conectar el módulo, de lo contrario no será capaz de reconocer el RTC y, por consiguiente, no podrá establecer la comunicación.

4.5.2 Configuración del RTC

La configuración del reloj no se lleva a cabo dentro de Python como en las secciones anteriores, se debe realizar desde la Terminal de la Raspberry.

El primer paso es corroborar que la Raspberry Pi es capaz de detectar el reloj, para ello se debe de ejecutar el comando *sudo i2cdetect -y 1*. Este comando hace un escaneo de todos los dispositivos que se encuentran conectados al bus I²C. El número que va después de la "*-y*" va a depender del modelo de Raspberry utilizado, en este caso se le coloca un 1.

```
pi@raspberrypi ~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figura 36. Muestreo de direcciones del DS1307.

Si se obtiene una lectura con el ID número 68 como el que se observa en la Figura 36, significa que el sensor se detectó correctamente y que se encuentra funcionando.

Para configurar la hora se necesita cargar el módulo RTC en el Kernel [17]de la Raspberry con la siguiente serie de comandos.

```
sudo modprobe rtc-ds1307
sudo bash
echo ds1307 0x68 &gt; /sys/class/i2c-adapter/i2c-1/new_device
exit
```

Ya se encuentra añadido el módulo, por lo que la Raspberry será capaz de comprobar la hora del sistema con la del RTC después de cada reinicio. Para sobrescribir la hora del sistema en el RTC se utiliza el comando *sudo hwclock -w*, ya que por defecto viene configurada al 1 de enero de 2000, y para leer la hora del RTC se ejecuta el comando sudo *hwclock -r*.

El segundo paso es lograr que la Raspberry utilice la hora del RTC por defecto. Para ello se necesita añadir el módulo kernel del RTC al fichero /etc/modules. Esto se realiza ejecutando el comando *sudo nano /etc/modules* y agregando la línea *rtc-ds1307* al término del fichero, como se muestra en la siguiente imagen.

[17]Kernel es la parte central o el corazón de un sistema operativo.

```
  GNU nano 2.7.4                Fichero: /etc/modules

# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.

i2c-dev
rtc-ds1307

                              [ 7 líneas leídas ]
^G Ver ayuda ^O Guardar    ^W Buscar     ^K Cortar txt^J Justificar^C Posición
^X Salir     ^R Leer fich.^\ Reemplazar^U Pegar txt ^T Ortografía^_ Ir a línea
```

Figura 37. Fichero /etc/modules.

Finalmente, se debe agregar el dispositivo DS1307 en el archivo de arranque rc.local. Para ello se ejecuta el comando *sudo nano /etc/rc.local* y se agregan las siguientes líneas.

```
echo ds1307 0x68 &gt; /sys/class/i2c-adapter/i2c-1/new_device
sudo hwclock -r
```

Las líneas se deben de añadir justo arriba del comando exit 0 como se muestra a continuación.

```
  GNU nano 2.7.4                    Fichero: /etc/rc.local

#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi
sudo modprobe rtc-ds1307
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
sudo hwclock -r
exit 0

                [ línea 21/23 (91%), col 1/16 (6%), car 496/519 (95%) ]
^G Ver ayuda ^O Guardar    ^W Buscar     ^K Cortar txt^J Justificar^C Posición
^X Salir     ^R Leer fich.^\ Reemplazar^U Pegar txt ^T Corrector ^_ Ir a línea
```

Figura 38. Fichero /etc/rc.local.

Con esto queda concluida la configuración del módulo RTC.

4.5.3 Análisis de Resultados

```
pi@raspberrypi:~ $ sudo hwclock -r
2018-07-20 14:10:29.392612+0200
pi@raspberrypi:~ $ date
vie jul 20 14:10:32 CEST 2018
```

Figura 39. Comparación de hora del sistema con la hora del RTC.

Para garantizar que la configuración del reloj en tiempo real se efectuó correctamente. Se necesita desconectar la Raspberry de la red y en seguida reiniciarla. Una vez hecho esto, se compara la hora del RTC con la hora del sistema mediante el comando date. Como se aprecia en la figura 39, las dos horas coinciden, así que ya no hay problema si la Raspberry llega a perder la conexión a internet.

4.6 Implementación del led

La incorporación del diodo led está pensada a modo de indicador visual. Para que este parpadee un número especifico de veces dependiendo de la acción que esté ejecutando el CAPTOR. Por mencionar algunos ejemplos, se usará para identificar cuando el nodo esté ejecutando una medición; cada vez que intente subir dichas mediciones al servidor; cada que tenga un error de conexión, entre muchos otros. Los leds se caracterizan por su bajo consumo de energía, debido a que no desperdician energía en calor para producir iluminación y por su larga vida útil.

4.6.1 Montaje en la Raspberry

En seguida, se enseña y explica a detalle el conexionado del diodo led.

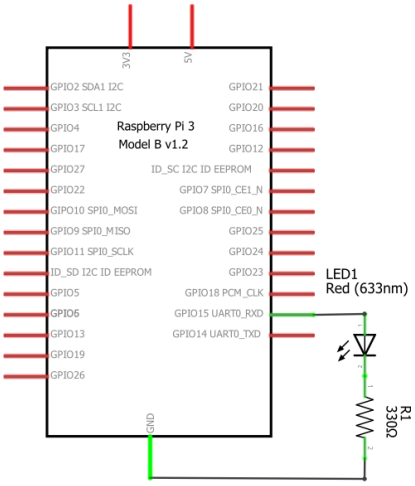


Figura 40. Diagrama esquemático del circuito.

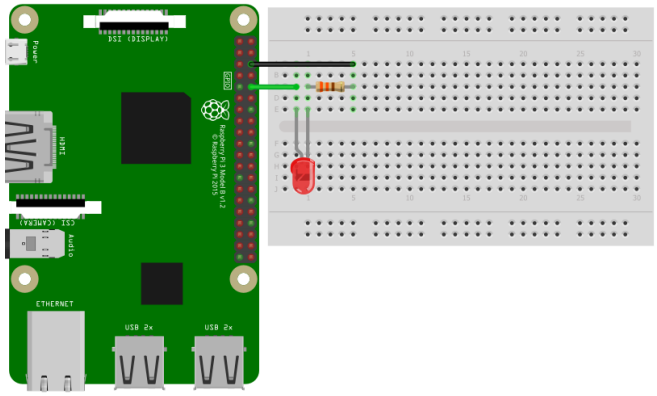


Figura 41. Circuito armado en Protoboard.

Es importante conectar el diodo led exactamente como se muestra en las Figuras 40 y 41. Al ser un diodo, únicamente permite el paso de corriente en

un solo sentido y, por consiguiente, solo emitirá luz cuando esté polarizado directamente.

Con el fin de limitar la corriente que pasa por el led, se coloca una resistencia en su terminal negativa. Esto se hace para no dañar el led o incluso la propia Raspberry

Para determinar el valor de la resistencia hay que considerar que la corriente que circula por el led debe de ser entre los 10 y 40 mA. Entre menor sea la corriente que circula a través de ellos, mayor será su eficiencia, así que únicamente se aplica la ley de Ohm para obtener el valor óptimo.

$$R = \frac{V}{I} = \frac{3.3\ V}{10\ mA} = 330\ Ohms$$

4.6.2 Funcionamiento y explicación del código Python

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(15, GPIO.OUT)
def blink(num):
        for i in range(0,num):
                GPIO.output(15, GPIO.HIGH)
                time.sleep(1)
                GPIO.output(15, GPIO.LOW)
                time.sleep(1)
while True:
        blink(4)
GPIO.cleanup()
```

Tabla 8. Código para prueba del led.

Como se puede notar en la Tabla 8, el código para la implementación de un diodo led es bastante sencillo. De igual manera que con el sensor DHT11, se necesita anexar la librería que permite trabajar con los pines GPIO. Se indica que se está trabajando con la numeración BCM y después se establece el pin GPIO 15 como salida.

Se crea la función blink que genera el número de parpadeos correspondientes al número que reciba. En la función simplemente se asigna un valor lógico alto para encender el led y otro valor lógico bajo para apagarlo, con un intervalo de un segundo de por medio.

En el ciclo infinito se manda a llamar a la función blink, estableciendo entre paréntesis el número de parpadeos deseados y finalmente se hace una limpieza de los pines GPIO utilizados.

4.7 Implementación del modem Huawei E303

La razón de incorporarle un modem al nodo CAPTOR es evidente, puesto que en la mayoría de las ocasiones va a estar funcionando en lugares donde no exista cobertura WiFi, ya sea en casas de voluntarios o en distintos edificios repartidos por la ciudad. El modem Huawei E303 es un dispositivo de internet móvil que trabaja con redes 2G, 3G, GPRS, GSM, EDGE, HSUPA Y HSDPA y, y soporta las bandas GSM de 850, 900, 1800, 1900 y 2100 MHz.

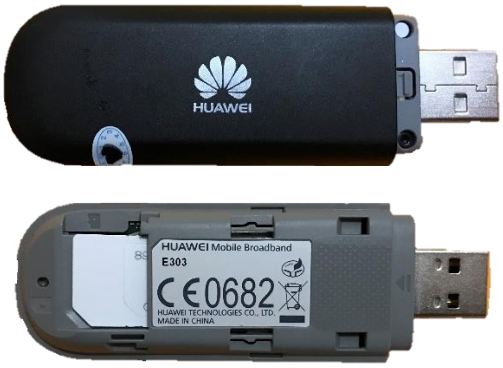


Figura 42. Vista frontal y trasera del Huawei E303.

La implementación del modem no resulta tan trivial como se pudiera imaginar. Una de las razones es que estos dispositivos incorporan a su vez una ranura de tarjeta microUSB. El problema radica en que la Raspberry no es capaz de

detectarlo como modem, más bien lo reconoce como un dispositivo de almacenamiento. Un factor para tomar en cuenta es que el modem consume bastante potencia, por eso se recomienda utilizar un alimentador que entregue bastante amperaje.

4.7.1 Configuración del modem

Lo primero es lograr que la Raspberry detecte el dispositivo como modem, se debe conectar el modem a la placa y ejecutar el comando *lsusb* para mostrar los dispositivos conectados a los puertos USB. Se puede ver que el modem tiene un ID de 12d1 : 1f01.

```
pi@raspberrypi:~ $ lsusb
Bus 001 Device 005: ID 0461:4e22 Primax Electronics, Ltd
Bus 001 Device 004: ID 046d:c315 Logitech, Inc. Classic Keyboard 200
Bus 001 Device 007: ID 12d1:1f01 Huawei Technologies Co., Ltd.
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp. SMSC9512/9514 Fast Ethernet Adapter
Bus 001 Device 002: ID 0424:9514 Standard Microsystems Corp. SMC9514 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Figura 43. Dispositivos conectados a la Raspberry PI.

Se debe instalar el programa usb_modeswitch, mediante el comando sudo *apt-get install usb-modeswitch.* Después se debe desconectar y volver a conectar el modem.

```
pi@raspberrypi:~ $ lsusb
Bus 001 Device 008: ID 0461:4e22 Primax Electronics, Ltd
Bus 001 Device 004: ID 046d:c315 Logitech, Inc. Classic Keyboard 200
Bus 001 Device 009: ID 12d1:14dc Huawei Technologies Co., Ltd.
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp. SMSC9512/9514 Fast Ethernet Adapter
Bus 001 Device 002: ID 0424:9514 Standard Microsystems Corp. SMC9514 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Figura 44. Dispositivos conectados a la Raspberry PI.

Si se vuelve a efectuar el comando *lsusb* se debería producir un cambio en el ID del modem, en este caso pasó de 2d1 : 1f01 a 2d1 : 14dc. Con esto, el cambio de modo tendría que realizar de manera automática.

El segundo paso es la configuración del modem. Se debe checar la IP del dispositivo Huawei mediante el comando *ifconfig*, después se tiene que copiar dicha IP en el navegador y se abrirá el sitio de configuración oficial Huawei. En este caso se introduce la IP de 192.168.8.1.

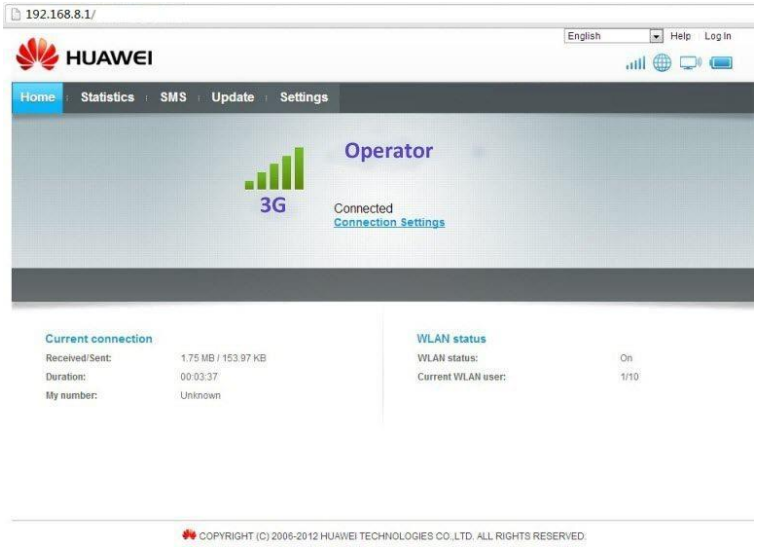


Figura 45. Sitio Web oficial Huawei.

Una vez en la web, se debe ir a la pestaña de Ajustes y finalmente activar el roaming. Una vez hecho esto, ya se debería tener conexión a internet. Se puede verificar ejecutando en la terminal el comando *ping google.es* y ver si se obtiene respuesta.

4.8 Implementación del código definitivo

Una vez que se ha adquirido el suficiente conocimiento de cada uno de los periféricos utilizados en el captor; una vez que se ha familiarizado con sus características técnicas, con su distribución de pines, con su funcionamiento interno, con su diagrama de conexiones; una vez que se ha probado cada circuito de manera independiente y verificado que todos ellos funcionan sin ningún error aparente, llega el momento de la integración. En el siguiente

diagrama, se muestra cómo queda la distribución del cableado de cada uno de los periféricos utilizados hasta el momento.

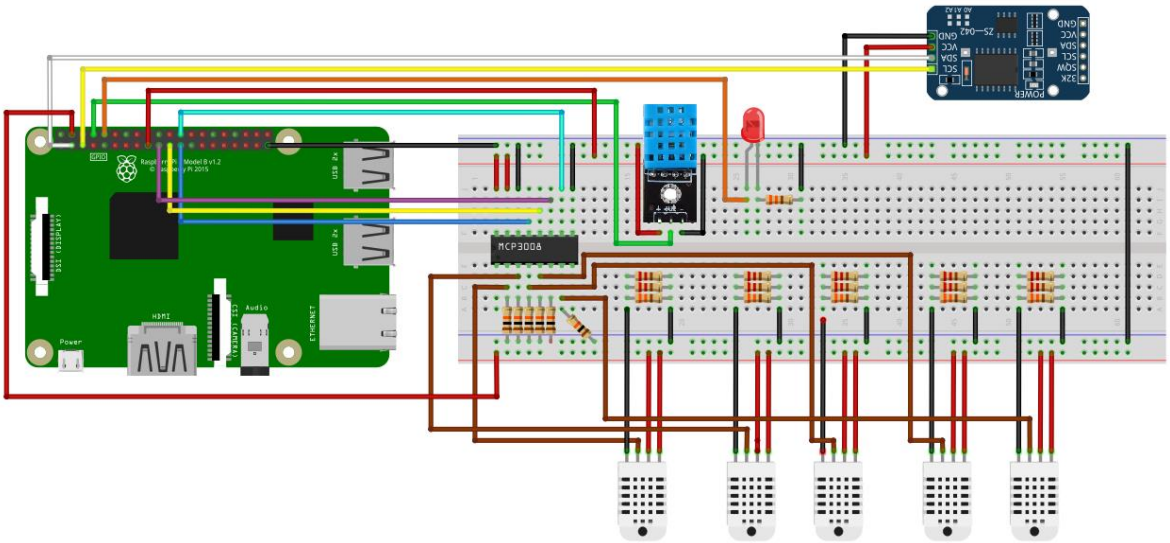


Figura 46. Diagrama de conexiones con todos periféricos.

Y aquí se muestran algunas imágenes de cómo queda implementado el prototipo en formato físico.

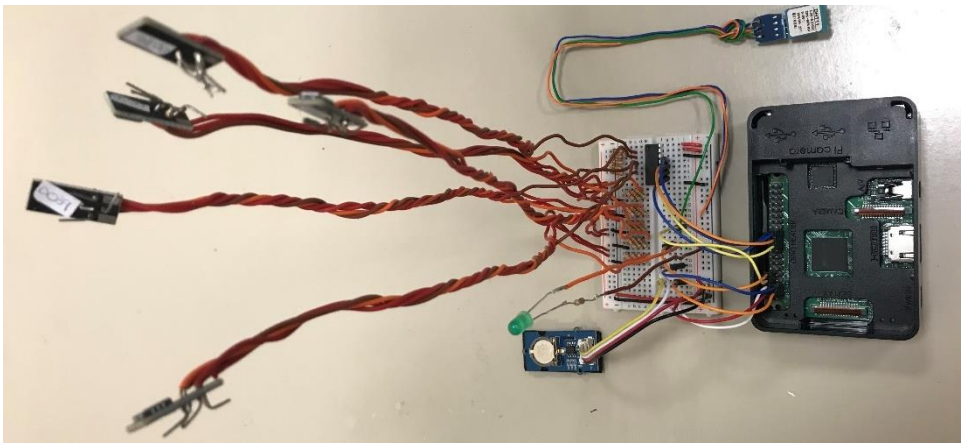


Figura 47. Implementación de los periféricos en físico.

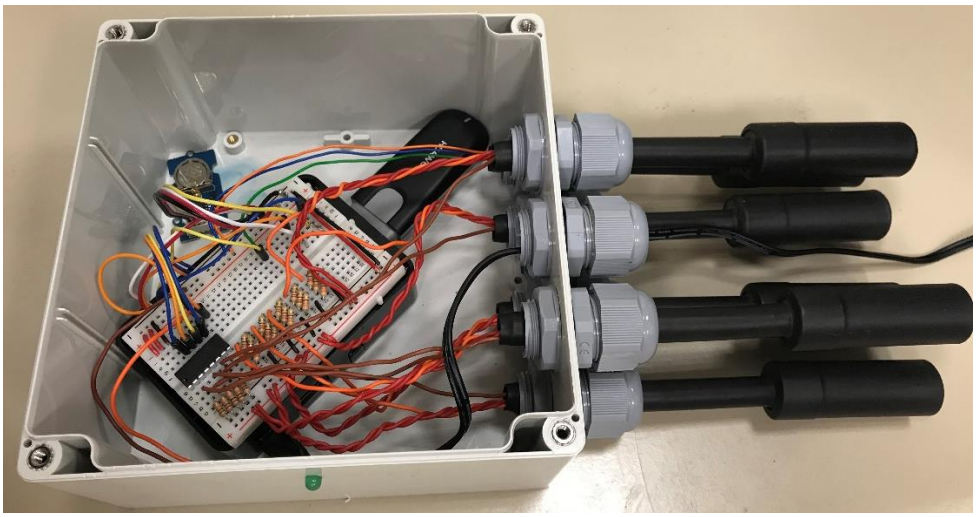


Figura 48. Diseño final del prototipo CAPTOR.

Como se mencionó anteriormente, los nodos CAPTOR que se encuentran operando actualmente, funcionan gracias a que combinan dos códigos fuente principales, uno desarrollado en Python y otro hecho en Sketch de Arduino. Un código fuente se define como el conjunto de líneas de texto en donde se dictan las instrucciones que tiene que seguir una computadora para ejecutar un programa.

La ventaja más destacable es la eliminación del bridge, el cual no es más que un cliente que permite la comunicación entre los dos códigos fuente y que comúnmente se utiliza para enviar los valores de los pines digitales y analógicos desde el Sketch de Arduino. Al tener que mandar a llamar los datos cada cierto tiempo, se da margen ante posibles errores de lectura y de comunicación, de hecho, gran parte del código Python está destinada a comprobar que este tipo de errores no sucedan. La siguiente ventaja se deriva del punto anterior, puesto que al eliminar el bridge también se quitarían bastantes líneas de código de comprobación de errores, lo que daría origen a un código mucho más compacto y limpio. Otra ventaja notable es que la unificación de ambos códigos significaría una reducción de la complejidad del

código, al no componerse por dos lenguajes de programación distintos. También implicaría que la manipulación o las futuras actualizaciones resulten mucho más sencillas.

Para empezar con la unificación del código, primer se debe de estudiar y analizar a detalle el funcionamiento de ambos códigos fuente. A este programa único de Python con Raspberry se le asignará el nombre de código definitivo. Por motivos de confidencialidad no es posible entrar a detalle en cada una de las líneas del código, así que únicamente se mostrarán pequeños fragmentos de código.

4.8.1 Código del Sketch de Arduino

El sketch de Arduino se compone de 4 métodos principales.

En el método *getSensorData* se van a adquirir los valores de los sensores MICS-2614 y del sensor DHT11.

```
void getSensorData(BridgeClient client) {
  dht.begin();
  float temperature= dht.readTemperature();
  float humidity= dht.readHumidity();
  float ozone1 = CS.analogRead2resistance(CS.readFilteredValue(O3_1,
NUMSAMLPLES, MINFILTER, MAXFILTER), CS.readVcc()/1000.,
RLOAD_MICS);
  // Print the values:
  //client.print(F("Current temperature: "));
  client.print(temperature);
  client.print("\t");
  //client.print(F("Current humidity: "));
  client.print(humidity);
  client.print("\t");
  //client.print("Current ozone (sensor 1): ");
  client.print(ozone1);
  client.print("\t");
```

Tabla 9. Método Get sensorData

Se puede ver que, para obtener los valores de temperatura y humedad, primero inicializa la librería y después llama a las funciones correspondientes de temperatura y humedad. Para obtener el valor de resistencia de los sensores de ozono, se utiliza un módulo llamado *analogRead2resistance* de la librería CS, que de igual forma aplica la ecuación del divisor de voltaje que se mencionó en el apartado 4.2.2.2. Debido a que todos los resultados se obtienen mediante un mismo método, se realiza la impresión del dato y la impresión de un \t. Ese \t representa una función específica, la cual se explicará más adelante en la siguiente sección.

El segundo y tercer método están relacionados al RTC. El método *getDate*, como su nombre lo indica, va a obtener la fecha del RTC y la va a sustituir por la temperatura del sistema, como se puede ver en la siguiente tabla. En este caso se muestra el módulo de la librería CS.

```
void CSApiHelper::getTime(char* time) {
RTC_DS1307 tempclock;//define a object of DS1307 class
  if (! tempclock.isrunning()) {
          tempclock.begin();
  }
  DateTime now = tempclock.now();
  sprintf(time, "%04d-%02d-%02dT%02d:%02d:%02dZ",now.year(),
now.month(), now.day(), now.hour(), now.minute(), now.second());
}
```

Tabla 10. Método getTime.

Y en el método setDate, se puede fijar manualmente una hora en específico en caso de que no se cuente con conexión a internet. Cabe mencionar que el código definitivo no necesita del empleo de estos dos métodos, ya que el RTC se configura de manera independiente en la Terminal.

El cuarto método es para realizar el parpadeo de los leds. De igual forma se muestra el módulo de la librería CS.

```
void CSApiHelper::blink(int times, int period){
   pinMode(13, OUTPUT);
   for (uint8_t i=0;i<times;i++){
     digitalWrite(13, HIGH);
     delay(period);
     digitalWrite(13, LOW);
     delay(period);
   }
```

Tabla 11. Método blink.

Se puede observar que la función tiene cierta similitud en comparación a la que se implementó con los GPIO de Raspberry, con pequeños cambios a la hora de activar y desactivar el pin digital.

Todos estos métodos tienen algo en común, y es que todos imprimen los valores médidos en el cliente o en la pipe. La pipe es un mecanismo que permite la comunicación y sincronización entre procesos de una misma máquina. El *client.print* no es más que un escritor de fichero.

4.8.2 Código Python

El funcionamiento del bucle principal del código Python es el siguiente. El nodo captor adquiere el valor de los sensores cada minuto y cada media hora sube los datos adquirido a un servidor CSV. Lo hace de esta manera debido a que realiza una media con los 30 datos acumulados y así obtener un valor más preciso. Las mediciones de cada uno de los nodos están desplazadas en el tiempo para que no todos suban a la misma hora.

Existen muchas líneas que se deben modificar para que el código sea capaz de correr satisfactoriamente en la Raspberry; sin embargo, las más importantes son todas aquellas líneas que llevan el nombre de *arduinoCall*, es evidente la razón por la que deben sustituirse.

La primera aparición de *arduinoCall* es justamente la propia función, tal y como se observa en la Tabla 12.

```
def arduinoCall(action):
try:
p = Popen(["curl", ARDUINO_URL + action], stdin=PIPE, stdout=PIPE,
stderr=PIPE)
output, err = p.communicate()
rc = p.returncode
if rc != 0: # Something went bad in the curl process
return 1
output = output.translate(None, "\r\n")
return output
except Exception as e:
with open(FILENAMELOG, "a") as logfile:
logfile.write("Something went wrong with Arduino call:\n")
print e
with open(FILENAMELOG, "a") as logfile:
logfile.write("%s\n" %(e.message))
return 1
```

Tabla 12. Función arduinoCall.

El objetivo principal de esta función es hacerle un curl[18] al Arduino mediante pipes y poder establecer la comunicación entre ambos programas. Se puede prescindir de esta función debido a que no se está trabajando con ningún Arduino. También se puede observar la comprobación de errores que se realiza en caso de que algo vaya mal con el Arduino. Hay que recordar que entre más líneas se eliminen, más sencillo se vuelve el programa.

```
r = arduinoCall("blink/5")
if r == 1:
   with open(FILENAMELOG, "a") as logfile:
      logfile.write("Couldn't perform init blink"
      "(This is likely to happen when Arduino and Linino are not synchronized)"
      "We will now reset the Arduino, please wait...")
```

[18]Curl es una librería de funciones que permite conectar con servidores y poder trabajar con ellos

```
subprocess.call(["reset-mcu"])
```

Tabla 13. Función arduinoCall con blink.

La segunda aparición de ArduinoCall que se puede observar en la Tabla 13, esta llamada se realiza para que el led parpadee 5 veces. De igual forma se observan más comprobaciones de errores, esta es otra clara ejemplificación de los posibles errores que puede producir el bridge.

Para realizar la misma funcionalidad en el código definitivo, se deben borrar absolutamente todas las líneas mostradas anteriormente y sustituirlo por la línea *blink(5)*.No existe punto de comparación en cuanto a sencillez se refiere.

```
dtime = arduinoCall("getDate")
if dtime == 1:
    with open(FILENAMELOG, "a") as logfile:
        logfile.write("Couldn't get date from RTC”)
        “We will now reset the Arduino, please wait”
    time.sleep(15)
```

Tabla 14. Función arduinoCall con getDate.

El uso de arduinoCall en esta tercera aparición sirve para leer la fecha del RTC y sustituirla por la temperatura del sistema. Se repite el mismo caso de no poder establecer la comunicación bridge.

En el código definitivo únicamente bastaría con sustituir todo el código mostrado por la línea *dtime = time.strftime("%Y-%m-%dT%H:%M:%SZ")*. El comando *strftime* permite devolver una cadena que representa la fecha, controlada por una cadena de formato explícita, que es justamente lo que se necesita.

```
output = arduinoCall("getSensors")
    if output == 1:
        with open(FILENAMELOG, "a") as logfile:
            logfile.write("Couldn't get sensors data")
            "We will now reset the Arduino, please wait"
        subprocess.call(["reset-mcu"])
        time.sleep(15)
    else:
        try:
            [temp, humid, o31, o32, o33, o34, o35] = output.split("\t")
            if DEBUG == True:
                with open(FILENAMELOG, "a") as logfile:
                    logfile.write("Sensor 1 value is " + o31 + "\n")
                with open(FILENAMELOG, "a") as logfile:
                    logfile.write("Sensor 2 value is " + o32 + "\n")
                with open(FILENAMELOG, "a") as logfile:
                    logfile.write("Sensor 3 value is " + o33 + "\n")
                with open(FILENAMELOG, "a") as logfile:
                    logfile.write("Sensor 4 value is " + o34 + "\n")
                with open(FILENAMELOG, "a") as logfile:
                    logfile.write("Sensor 5 value is " + o35 + "\n")
                with open(FILENAMELOG, "a") as logfile:
                    logfile.write("Temp value is " + temp + "\n")
                with open(FILENAMELOG, "a") as logfile:
                    logfile.write("Humidity value is " + humid + "\n")
```

Tabla 15. Función arduinoCall con getSensors.

Finalmente, en esta aparición se puede ver como se reciben los datos de resistencia de los distintos sensores. Ahora se continuará con la explicación del comando \t, ejemplificándolo con el código de la Tabla 15. Anteriormente, se dijo que el getSensorData obtenía todos los resultados dentro del mismo método, imprimiendo también un \t para cada sensor. Resulta que ese \t es el carácter de parseo que tiene Python. Gracias a la incorporación de ese \t se hace posible el separar los valores de los sensores, cuando se ejecuta el comando *output.split("\t")* en el codigo Python.

La sustitución de esta funcionalidad en el código definitivo quedaría de la siguiente forma.

```
with open(FILENAMELOG, "a") as logfile:
    logfile.write("Sensor 1 value is " + str(o31))
    print("O3,1 Sensor:  {} ({} KOhms)".format(o31volts,o31))
with open(FILENAMELOG, "a") as logfile:
    logfile.write("Sensor 2 value is " + str(o32))
    print("O3,2 Sensor:  {} ({} KOhms)".format(o32volts,o32))
with open(FILENAMELOG, "a") as logfile:
    logfile.write("Sensor 3 value is " + str(o33))
    print("O3,3 Sensor:  {} ({} KOhms)".format(o33volts,o33))
with open(FILENAMELOG, "a") as logfile:
    logfile.write("Sensor 4 value is " + str(o34))
    print("O3,4 Sensor:  {} ({} KOhms)".format(o34volts,o34))
with open(FILENAMELOG, "a") as logfile:
    logfile.write("Sensor 5 value is " + str(o35))
    print("O3,5 Sensor:  {} ({} KOhms)".format(o35volts,o35))
with open(FILENAMELOG, "a") as logfile:
    logfile.write("Temp value is " + str(temp))
    print("Temperature:  {} C".format(temp))
with open(FILENAMELOG, "a") as logfile:
    logfile.write("Humidity value is " + str(humid))
    print("Humidity:     {} %".format(humid))
```

Tabla 16. Código sin arduinoCall.

Únicamente hay que llamar a las funciones e imprimir el dato. En este caso no se necesita separar el valor de los sensores, debido a que desde un inicio se leyeron cada uno por separado.

Existen más apariciones de arduinoCall a lo largo del programa; no obstante, la funcionalidad es la misma que ya se explicó en las apariciones anteriores.

4.9 Ejecución del código definitivo

El código definitivo ya casi está listo para poder ejecutarse. En realidad, ya se puede correr, solo que no será capaz de subir los catos CSV al servidor de CommSensum, debido a que aún no se ha dado de alta el nodo en el servidor.

CommSensum es un servidor creado por el grupo de investigación SANS destinado a trabajar con proyectos orientados al internet de las cosas. En este servidor se dan de alta los nodos Captor y se obtiene un enlace en el que se suben los datos para que posteriormente queden registrados.

Antes que nada, necesitará generar una llave de Autentificación SSH. Estas llaves se caracterizan por venir en pares; una clave privada y una pública. Por lo general, la clave privada se guarda como *~/.ssh/id_<type>* y la clave pública es *~/.ssh/id_<type>.pub*. El tipo de cifrado más utilizado es RSA, por lo que sus claves deben tener el nombre *id_rsa* y *id_rsa.pub*. La clave pública debe entregarse libremente y agregarse a los servidores a los que se desea conectarse en el archivo ~/.ssh/authorized_keys. La clave privada debe estar protegida en su máquina local con reglas de acceso estrictas.

Para crear la llave RSA se deben colocar los siguientes comandos en la Terminal de la Raspberry.

```
mkdir -p ~/.ssh
sudo apt-get install dropbear
dropbearkey -t rsa -f ~/.ssh/id_rsa
dropbearkey -y -f ~/.ssh/id_rsa | grep "^ssh-rsa" > authorized_keys
```

Posteriormente, se debe tener acceso a un ordenador o PC local, para poder conectarse al servidor de CommSensum. Se debe introducir la línea ssh csv@csv.sans-ac-upc.org -p 333 en una Terminal, con su respectiva contraseña. Después se deben ejecutar los siguientes comandos.

```
sudo -s
CAPTOR_ID=20003
mkdir cd /home/jail/home/captor${CAPTOR_ID}
cd /home/jail/home/captor${CAPTOR_ID}
mkdir .ssh
mv ~/auth_${CAPTOR_ID} .ssh/
cd .ssh && cat auth_${CAPTOR_ID} >> authorized_keys
```

```
cd ..
```

Con esta serie de comandos se da de alta el ID deseado y se mueve la llave recién creada a las llaves autorizadas del servidor. Una vez hecho esto, el nodo ya debería de ser capaz de enviarle datos a CommSensum.

4.9.1 Análisis de resultados

```
csv@csvserver:~/captor-data$ tail -n 20 captor20003.csv
2018-07-20T20:30:26;17.0737;7.3070;3.9713;4.0290;6.4023;21.47;26.07;14.9144;2017
/06/16
2018-07-20T21:00:30;17.5017;7.3790;4.1187;4.1080;6.6960;23.33;28.53;15.3635;2017
/06/16
2018-07-20T21:30:34;18.2420;7.4330;4.1987;4.2973;7.2557;22.40;27.53;15.1736;2017
/06/16
2018-07-20T22:00:37;19.8410;7.5290;4.2567;4.4573;7.8927;24.27;29.50;15.8037;2017
/06/16
2018-07-20T22:30:41;20.9420;7.6790;4.2920;4.3960;7.6470;27.07;32.87;16.4936;2017
/06/16
2018-07-20T23:00:45;21.5963;7.8037;4.3507;4.4497;7.7337;25.20;30.60;16.0458;2017
/06/16
2018-07-21T00:30:25;22.3013;8.0450;4.5250;4.9903;10.0687;24.23;29.47;16.0311;201
7/06/16
2018-07-21T01:00:30;23.2500;8.0927;4.5977;4.6813;9.1333;20.53;24.97;14.9601;2017
/06/16
2018-07-21T01:30:33;24.0723;8.1967;4.5370;4.5087;8.4327;23.33;28.40;15.5810;2017
/06/16
2018-07-21T02:00:37;22.9683;8.1823;4.6577;4.4507;8.1873;25.20;31.30;15.8435;2017
/06/16
2018-07-21T02:30:41;22.3670;8.4470;4.6443;4.6143;8.4470;27.07;33.83;16.3112;2017
/06/16
2018-07-21T03:00:45;22.2520;9.5167;4.7927;4.8310;9.6820;21.23;26.83;14.9094;2017
```

Figura 49. Datos del archivo CSV.

Se observa que el código definitivo es capaz de mandar el archivo CSV al servidor de CommSensum con éxito. También se aprecia que, por el momento, no presenta retrasos en sus mediciones, ya que todos los valores se han mandado en un intervalo de 30 minutos casi exactos.

Sí se quisiera analizar la precisión de los datos en comparación al ozono, cabe mencionar que de momento sería imposible. Cuando se construye un nodo CAPTOR desde cero, como en este caso. Es necesario llevarlo primero a calibrar a una estación de referencia por unas cuantas semanas, hasta obtener

un archivo de betas que servirán para calibrar los valores de resistencia que arroja el sensor.

Dicho de otro modo, los sensores actualmente están entregando un valor ROW; es decir, sin aplicar las betas, y darán otro resultado completamente diferente al aplicarle las betas, ese será el valor de concentración de ozono que se está buscando, expresado en $\frac{\mu g}{m^3}$.

5
Costes

En esta sección se desglosan todos los costes relacionados al proyecto. En la sección 5.1 se hace el presupuesto de todo el Hardware, en la sección 5.2 se analizan los costos de recursos humanos, en la sección 5.3 se mencionan los costos indirectos. Y en la sección 5.4 se hace un resumen de todos los costes.

5.1 Hardware

En seguida, se desglosan todos los elementos que componen a este prototipo CAPTOR, señalando el coste unitario, el número de unidades a adquirir, el costo total y el servidor correspondiente para comprarlos.

DESCRIPCIÓN	UNIDADES	PRECIO/ UNIDAD (€)	PRECIO TOTAL(€)	PROVEEDOR
Raspberry Pi 3 Model B+ Starter Kit	1	57.76	57.76	raspipc
ADC MCP3008	1	1.92	1.92	mouser
DHT11	1	4.25	4.25	mouser
RTC DS1307 con batería incluida	1	7.09	7.09	ebay
Huawei E303	1	51.69	51.69	amazon
Paquete de 10 MICS 2614	1	25.64	25.64	amazon
Protoboard	1	2.49	2.49	ebay
Paquete de 100 resistencias 10k 1% (sensor load)	1	1.21	1.21	ondaradio
Pack 100 resistors 3Paquete de 330 resistencias 10k 1% (sensor load)	1	1.21	1.21	ondaradio
Paquete de 100 resistencias 220 1% (sensor load)	1	1.21	1.21	ondaradio
Paquete de 40 jumpers macho hembra	1	2.1	2.1	ebay
Paquete de 40 jumpers macho macho	1	2.1	2.1	ebay
Tubo termoretractil de 19,1 mm	6	1.92	11.52	ondaradio
Tubo termoretractil de 25.4 mm	6	1.92	11.52	ondaradio
Caja de 160x160x90	1	29.95	29.95	Farnell
Prensaestopas M20 con tuerca	6	7.31	43.86	Farnell
Prensaestopas M20 con tuerca para la fuente	1	1.37	1.37	Farnell
			256.89	

Tabla 17. Conteo unitario del prototipo.

5.2 Personal

Únicamente se están tomando en cuenta los costos relacionados a la implementación de este nuevo prototipo CAPTOR. En seguida, se señalan las horas que asignan en cada fase y el precio que se cobra por hora. Se toma a 13 euros la hora, que es básicamente lo que cobran los juniors informáticos actualmente.

Fase	Horas	Precio(€/hora)	Subtotal (€)
Diseño y planificación	60	13	780 €
Implementar prototipo	70	13	910 €
Documentación	40	13	520 €
			2,210 €

Tabla 18: Costes y número de horas

Al diseño y planificación se le está relacionado todo el proceso de aprendizaje y familiarización del funcionamiento de los periféricos.

5.3 Costes indirectos

En esta sección se puede situar al gasto de la oficina de trabajo, la cual tiene capacidad para cinco personas y tiene un coste aproximado de 500€ al mes. Así que, el coste indirecto es de 100€ por cada mes o 500€ cada 5 meses, sumando un total de 500€. Los servicios básicos de luz, internet y agua ya van incluidos en el precio.

5.4 Resumen de Costes

Personal	Hardware	Indirectos	Total
2210€	304.3€	500€	3014.3€

Tabla 20. Resumen de costes.

Realizando la suma de todos los costes, se tiene que la actualización del CAPTOR tiene un costo de 3014.3 €.

6

Conclusion y Trabajo Futuro

Tras la realización del presente proyecto se ha logrado efectuar un análisis de los elementos que han incidido en él.

En primer lugar, el microcomputador responsable de llevar todo a cabo, la Raspberry Pi. Demostró ser un dispositivo superior a su contraparte Arduino Yun, al no haber crasheado ni una sola vez a lo largo de todo el proceso de aprendizaje. Esto sin mencionar, su mejor potencia de cálculo, sus capacidades multimedia y su costo inferior. Realmente las posibilidades que ofrece esta placa son ilimitadas.

Por otro lado, se tiene al lenguaje de programación Python. Como se comentó en los primeros apartando del proyecto, ha resultado ser un lenguaje sumamente intuitivo y simple de aprender; además de la practicidad que supone el uso de este lenguaje en aplicaciones este tipo.

El averiguar el funcionamiento de los periféricos supuso un desafío; no obstante, se logró incorporar todos y cada uno de ellos de manera satisfactoria.

Finalmente, la unificación de los códigos fuente, representó el mayor desafío. Al ser un código que ha sido modificado y manipulado por varias personas, se tuvieron que realizar conjeturas sobre el funcionamiento de algunas partes, conjeturas que al final resultaron ser ciertas.

La incorporación de todos estos elementos dio como resultado el nuevo prototipo CAPTOR, un nodo con un único código fuente, que se caracteriza por ser más limpio y más sencillo, y que también se tiene la certeza que su procesador interno va a subsistir en el mercado muchos años más.

Como trabajo futuro se puede señalar la incorporación de más elementos, aprovechando la potencia extra que ofrece la Raspberry Pi. La propuesta más factible sería la implementación de una veleta, para que el nodo sea capaz de obtener la dirección del viento, esta aplicación actualmente se encuentra incorporada en un único nodo captor. Es una opción perfectamente viable, ya que este dispositivo utiliza dos pines para el envío de datos, uno digital y otro analógico, lo que nos dejaría con dos puertos analógicos libres para más actualizaciones.

7

Bibliografía

- Bordachar, Andrés (2017). UDD. Proyecto HIRI. Fecha de consulta: Julio del 2018. Recuperado de http://www.udd.cl/noticias/2017/07/31/hiri-waze-la-contaminacion-ambiental-creado-ingenieria-udd/
- Mark Lutz. (2009). Learning Python. Estados Unidos: O'Reilly Media.
- Aqicn. MiCS-2614 Datasheet. SGX Sensortech, Courtils 1 CH-2035 Corcelles-Cormondrèche. 2009.
- Alarcón, Diego. (2015) Internet móvil con el modem 3G USB Huawei E303. Fecha de consulta: Julio del 2018. Recuperado de http://carlini.es/internet-movil-con-el-modem-3g-usb-huawei-e303-en-la-raspberry-pi/
- Solís, Miguel (2017). El ozono troposférico es el responsable de 1.800 muertes cada año en España. Fecha de consulta: Julio del 2018. Recuperado de

http://www.cronicanorte.es/%E2%80%8Be%E2%80%8Bl-ozono troposferico%E2%80%8B%E2%80%8B-responsable-1-800-muertes-ano-espana/113063.

- Navarro, Miguel Ángel (2018). Ecoticias. Proyecto de la UPCT. Fecha de consulta: Julio del 2018. Recuperado de https://www.ecoticias.com/co2/185910/proyecto-UPCT-mide-contaminacion-calles-escoge-rutas-limpias
- García, Vicente (2012). Introducción al I2C BUS. Fecha de consulta: Julio del 2018. Recuperado de https://www.diarioelectronicohoy.com/blog/introduccion-al-i2c-bus
- Llamas, Luis (2016). Reloj y calendario en Arduino con los RTC DS1307 y DS3231. Fecha de consulta: Julio del 2018. Recuperado de http://www.instructables.com/id/Raspberry-Pi-I2C-Python/
- Maxim Integrated. DS1307 datasheet. Maxim Integrated Products, Inc. Sunnyvale, CA. 2015.
- Mouser. DHT11 Humidity & Temperature Sensor datasheet. Mouser Electronics. 2016
- Raspberry Pi 3 Model B Specifications. Fecha de consulta: Julio del 2018. Recuperado de https://www.raspberrypi.org/products/raspberry-pi-3-modelb/
- Downey, A. (2015), Think Python. How to Think Like a Computer Scientist, Needham, Massachusetts: Green Tea Press.
- Croston, Ben. Raspberry-gpio-python. RPi.GPIO module basics. Fecha de consulta: Julio del 2018. Recuperado de https://sourceforge.net/p/raspberry-gpio-python/wiki/BasicUsage/

Lista de figuras

Lista de tablas