ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Trabajo Fin de Grado

Grado en Ingeniería de Sistemas de telecomunicación

Implementación de una estación meteorológica para el telescopio robótico de la UPCT

AUTOR: Laura Zamora Navarro
DIRECTOR: Antonio Pérez Garrido
Octubre / 2015

Índice

Índice de figuras

1. Introducción

Este proyecto consiste en el desarrollo y programación de la que será la estación meteorológica del telescopio robótico que se encuentra en el edificio de I+D de la Universidad Politécnica de Cartagena.

Consta de un sensor que mide la temperatura y la humedad del ambiente y de un anemómetro para obtener la velocidad del viento, los cuales poseerán unos valores de alarma en caso de que la humedad sea demasiado alta, la temperatura baje del punto de rocío o la velocidad del viento sea demasiado alta para llevar a cabo observaciones con seguridad. Además, como elemento innovador, poseerá una webcam que mediante fotografías al cielo determinará si se puede observar o no, gracias a una detección de las nubes mediante un análisis del color de las imágenes. Aunque esta parte se ha iniciado en este proyecto su finalización se pospone para un futuro debido a su complejidad.

Todo esto se integrará y programará mediante una Raspberry Pi, un pequeño ordenador de bajo presupuesto, utilizando como lenguaje de programación Python. La Raspberry Pi irá conectada al ordenador del telescopio, con el cual podrá comunicarse a través de un protocolo TCP/IP.

Durante esta memoria se irá detallando paso a paso cada elemento mencionado anteriormente y cómo se ha conseguido introducirlos en un solo programa que se ejecutará desde el ordenador del telescopio hasta conseguir el resultado deseado: una pequeña estación meteorológica a la que los ordenadores de control del telescopio podrán acceder remotamente para conocer las condiciones ambientales y decidir si es seguro llevar a cabo las observaciones, proceso que se realiza sin intervención humana.

2. Justificación del tema elegido

Durante el segundo cuatrimestre de cuarto de grado cursé la asignatura optativa de Ampliación de Física, en la cual uno de los profesores que impartió clase fue Antonio Pérez Garrido. Antonio nos propuso varios proyectos a un compañero y a mí, entre los cuales se encontraba el de la estación meteorológica. Después de un tiempo viendo las propuestas de otros profesores o los proyectos escogidos por otros compañeros me pareció que este era el más interesante.

Elegí este proyecto por el reto que sabía que iba a suponerme: tenía que aprender un lenguaje de programación completamente nuevo (Python), defenderme en un sistema operativo que tampoco había usado nunca (Linux/Debian) y, en definitiva, aprender de manera autodidacta sobre muchos temas, saber dónde buscar la información necesaria y valérmelas por mí misma ante situaciones complicadas.

En definitiva, me pareció un proyecto perfecto para demostrar que estoy preparada para afrontar cualquier problema, es decir, ejercer de ingeniera.

3. Objetivos

El objetivo principal de este proyecto es la construcción y programación de una estación meteorológica que sea capaz de llevar a cabo los siguientes puntos:

- Poder determinar con certeza si se puede observar el cielo con el telescopio con respecto a la humedad relativa, la velocidad del viento y las nubes que pueda haber. Debido a que el telescopio realiza sus tareas sin supervisión humana, es necesario que, ante condiciones inhóspitas que pueden dañar el instrumental, sea capaz de cesar la observación y cerrar la cúpula automáticamente.

- Dar un valor interactivo de la velocidad máxima del viento aceptable, además de otros parámetros que controlan cómo se realiza el cálculo de las rachas y la velocidad media. temperatura y humedad del ambiente.

- Obtener valores fiables de la temperatura y humedad ambiental en el exterior de la cúpula.

- Conseguir la compatibilidad de datos de los elementos anteriores con el programa ya existente que controla el telescopio y los mecanismos de apertura y cierre de las compuertas de la cúpula.

- La medida de la presión atmosférica suele incorporarse a las estaciones meteorológicas. Debido a que en las instalaciones telescópicas los valores de la presión no determinan si es posible llevar las observaciones a cabo o no, se decidió prescindir de ella.

4. Descripción del telescopio robótico

En este apartado detallaremos el estado actual del telescopio. El telescopio robótico se encuentra conectado directamente a un ordenador con sistema operativo Linux por medio de una IP.

Dado que el software que posee el motor de la cúpula no es compatible con Linux, hay otro ordenador con Windows que lo controla. Este ordenador a su vez está conectado al ordenador principal..

El objetivo es conectar la Raspberry Pi al ordenador con Linux, ya que es el único que tiene Internet, mediante el protocolo TCP/IP asignándole una IP fija, y emplearla como estación meteorológica, reemplazando la antigua. En caso de que el tiempo no fuese favorable para realizar observaciones, el ordenador con Linux mandaría un mensaje al ordenador con Windows para cerrar la cúpula.

A continuación podemos ver un esquema simple de la comunicación entre los diferentes elementos.

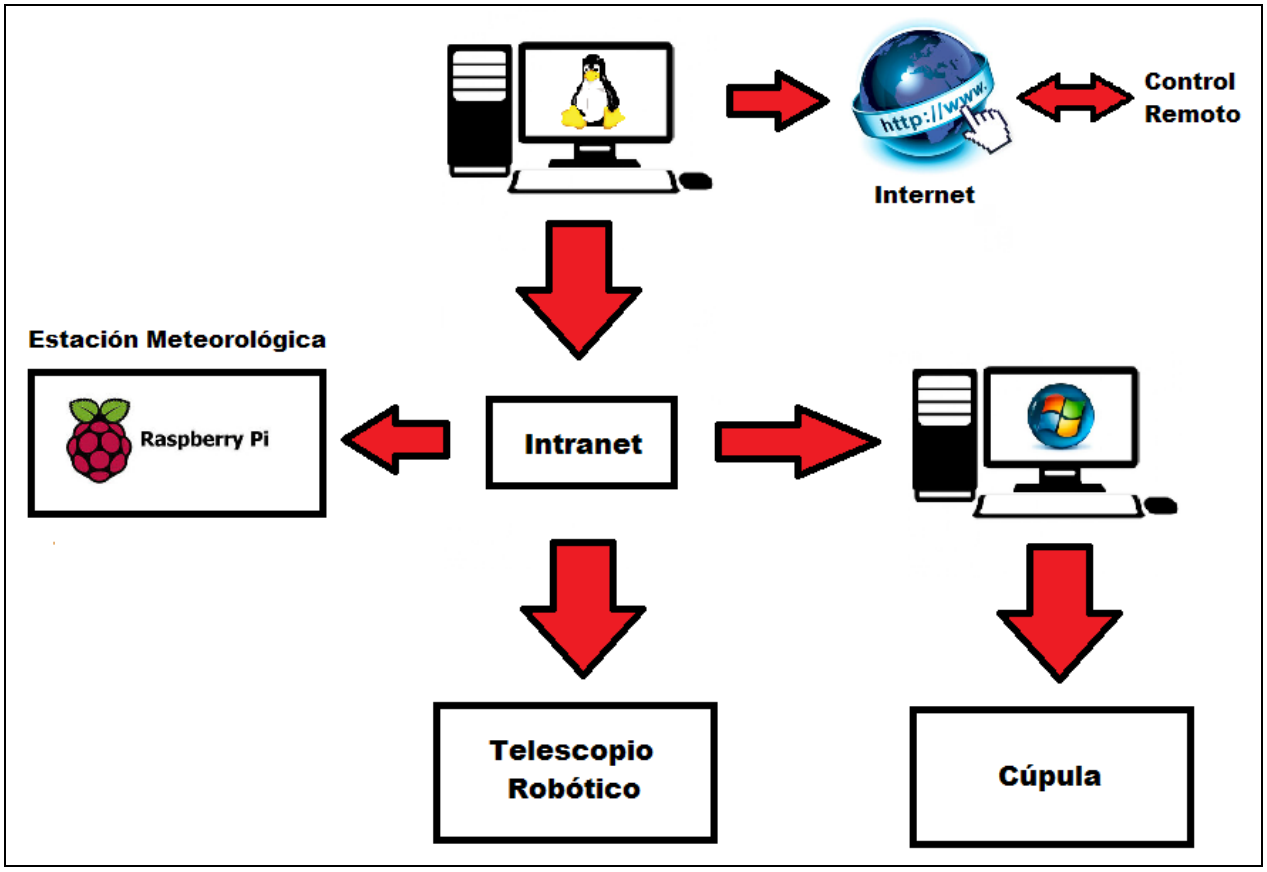


Ilustración 1 – Esquema del comunicación del telescopio

5. Metodología

El método utilizado para la realización del proyecto ha sido la búsqueda de información sobre el tema u objetivo marcado y la realización de este tras la recopilación y valoración de dicha información. El medio principalmente utilizado para la búsqueda de información ha sido Internet, y la resolución de dudas ha sido posible mediante el tutor, consulta de documentos técnicos o foros específicos sobre tecnología relacionada con la Raspberry y uso de sensores.

Primeramente me marqué una lista de objetivos a cumplir en un plazo de tiempo; en caso de no poder avanzar en uno de ellos podía pasar a otro sin obstaculizarme la progresión. Una vez creada la lista empecé a buscar información para el primer fin marcado, y después de poder contrastarla la ponía en práctica.

Debido al carácter autodidacta de esta técnica, me he basado en el método de "Ensayo y error", en el cual para conseguir un fin investigaba y probaba distintas maneras hasta conseguir el resultado deseado. De esta forma he aprendido a alcanzar las metas a base de equivocarme muchas veces.

5.1. Raspberry Pi vs Arduino

En el proceso de selección de la plataforma física que iba a soportar la estación meteorológica se pensó en dos posibles candidatos: la Raspberry Pi y el Arduino. El precio y el tamaño de los dos dispositivos son comparables; ya sabíamos que Raspberry Pi y Arduino eran pequeñas y baratas. Lo que hay en su interior es lo que las distingue.

Para empezar, Raspberry Pi es un "ordenador low-cost" completamente funcional, mientras que Arduino es un microcontrolador, el cual es sólo un componente. Aunque el Arduino puede ser programado con pequeñas aplicaciones en un lenguaje similar al C, este no puede ejecutar todo un sistema operativo. La Raspberry Pi es 40 veces más rápida que un Arduino cuando se trata de velocidad de reloj. Además, Pi tiene 128.000 veces más memoria RAM. La Raspberry Pi es un ordenador independiente que puede ejecutar un sistema operativo real en Linux. Puede realizar varias tareas, soportar dos puertos USB y conectarse a Internet de forma inalámbrica o mediante un puerto ethernet.

Puede sonar que Raspberry Pi es superior a Arduino, pero eso es sólo cuando se trata de aplicaciones de software. La simplicidad de Arduino hace que éste sea una apuesta mucho mejor para proyectos de hardware. Arduino tiene la capacidad "analógica" y en "tiempo real" que la Pi no: esta flexibilidad le permite trabajar con casi cualquier tipo de sensor o chip.

Finalmente se escogió la RPi debido a que tiene más prestaciones que Arduino, aunque este funcione mejor con los sensores, ya que era necesario crear un programa complejo que pudiese conectarse a una red Ethernet para enviar y recibirdatos vía TCP/IP.

5.2. Materiales

Los recursos físicos utilizados para el proyecto han sido:

- Ordenador.
- Raspberry Pi.
- Conversor A/D para la Raspberry Pi.
- Anemómetro.
- Sensor de temperatura y humedad.
- Adaptador USB de WiFi.
- Cámara Web.
- Cable HDMI.
- Pantalla con entrada HDMI.
- Teclado.
- Ratón.
- Fuente de alimentación de 9V / 1A fija.
- Fuente de alimentación con cable micro USB de 5V /1A.

5.3. Velocidad del viento

Un sensor que determine la velocidad del viento es imprescindible en un telescopio robótico. A partir de ciertas velocidades del viento no es conveniente abrir la cúpula del telescopio. Un viento excesivo puede dañar las compuertas e incluso levantar la cúpula de su emplazamiento. Para ello se buscó un anemómetro que fuese compatible con la Raspberry Pi.

A la hora de escoger el anemómetro me encontré básicamente con dos tipos: el primero de ellos era muy barato y simple, donde para determinar la velocidad del viento había que hacer un cálculo de cuántas vueltas daba el aparato por segundo (rotación por unidad de tiempo).

Ilustración 2 – Anemómetro de Maplin "low-cost"

El segundo tipo, y el que finalmente escogí, era bastante más caro pero me gustó mucho más su funcionamiento. Dependiendo de la velocidad a la que giren las aspas, el anemómetro da una salida de voltaje analógica, de entre 0.4V (equivalente a 0 m/s) a 2.0V (equivalente a 32 m/s).

Ilustración 3 – Anemómetro con salida analógica de Adafruit

El problema que presentaba este sensor es que la Raspberry Pi no tiene entradas analógicas, por tanto hubo que incorporar un conversor analógico/digital, el MCP3008.

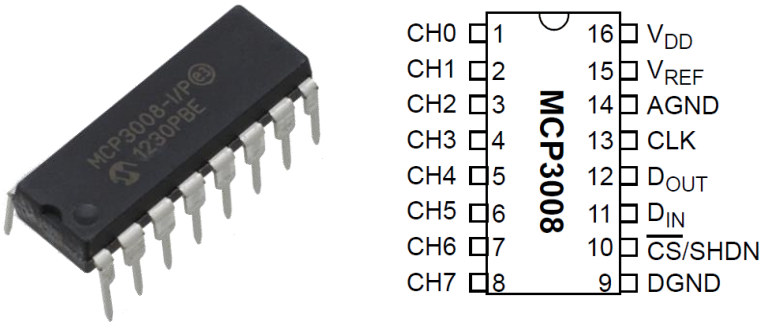


Ilustración 4 – Conversor A/D MCP3008

Conectando el anemómetro correctamente a la Raspberry, como detallaremos en la sección de desarrollo, y con el código diseñado correspondiente, el sensor funciona con precisión dando valores de salida del viento con la periodicidad establecida en el software, y emitiendo una señal de alarma si se supera el valor máximo de viento (como valor inicial marcaremos los 8 m/s, pero que puede ser modificado por el usuario).

5.4. La temperatura y la humedad del ambiente

Para medir la temperatura y la humedad se buscó un sensor o sensores compatibles con la Raspberry Pi que fuesen económicos y a la vez pudiesen colocarse en el exterior sin sufrir daños cuando hubieran condiciones adversas.

El motivo por el que se necesita conocer la temperatura y humedad es la posible condensación del agua de la atmósfera cuando la temperatura es inferior al punto de rocío, lo que dañaría el telescopio y cualquier otro instrumento en el interior de la cúpula. El objetivo de nuestro sensor será monitorizar la temperatura y humedad del ambiente para calcular la temperatura de rocío, de manera que si esta es inferior a la temperatura actual se informe a los ordenadores de control del telescopio que se debe cerrar la cúpula inmediatamente.

Primeramente se consideró un sensor con salida de datos GPIO, compatible con la Raspberry, que medía tanto la humedad como la temperatura: el Aosong AM2302, el cual fue escogido por su económico precio, su amplio rango de valores y por tener las salidas cableadas, al contrario de sus dos modelos más similares: DHT11 y DHT22.

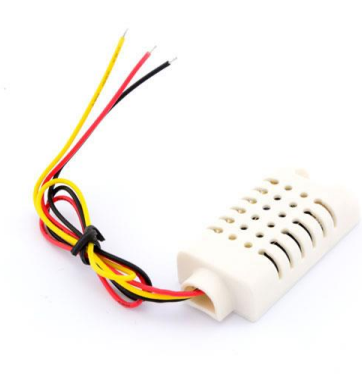


Ilustración 5 – Sensor de temperatura y humedad AM2302

Finalmente, después de diversas pruebas con varias librerías y códigos comprobé que no era totalmente compatible con la Raspberry. Por este motivo descartamos usar este sensor. Ahondando más en el tema encontré también que el AM2302 tenía muchos problemas con la RPi a la hora de dar datos fiables, teniendo un porcentaje de error en las lecturas de un 40%. El sensor que se escogió para sustituirlo es el Aosong AM2315.

El AM2315, a diferencia del AM2302, tiene como i2c como interfaz de salida de datos, lo que le da una mayor fiabilidad en las lecturas, además de estar mejor preparado para colocarse en exteriores. Este sensor, como podemos ver más detalladamente en el anexo, posee un sensor capacitivo para determinar la humedad y varios dispositivos integrados de alta precisión para medición de la temperatura, conectados ambos a un microprocesador.

Ilustración 6 – Sensor de temperatura y humedad AM2315

Una vez instaladas las librerías necesarias para la configuración de la interfaz i2c, cableado correctamente el sensor y realizado el código de lectura, el sensor ha resultado funcionar sin ningún error en todas las simulaciones realizadas y dando valores de temperatura y humedad contrastadas con los arrojados por otros termómetros e hidrómetros.

5.5. Detección de la nubosidad del cielo

Aún a pesar de que el viento, la humedad y la temperatura sean óptimas para realizar observaciones el cielo puede estar cubierto de nubes por lo que sería inútil abrir la cúpula, además del riesgo que supone una lluvia repentina sobre el telescopio y el resto de instrumental.

Primeramente se plantearon dos opciones para la detección de las nubes: la primera y más sencilla era colocar un sensor infrarrojo apuntando al cielo, que dependiendo de la temperatura que midiese (si había un incremento de temperatura) se podría saber si hay nubes o no, pero este método era un poco pobre en sí, ya que el sensor sólo puede detectar lo que tiene estrictamente encima suya. Se pensó que para complementar esta información se podía extraer información de la web de la Agencia Estatal de Meteorología (AEMet) mediante una herramienta que nos muestra la probabilidad de precipitación, y por tanto de nubes, en Cartagena.

Como segunda opción pensamos en instalar en la cúpula del telescopio una cámara web con un objetivo gran angular con la cual haríamos fotos al cielo y tras un procesado de la imagen podríamos detectar si hay nubes o no. Finalmente nos decantamos por la segunda opción por ser más completa y novedosa.

Dado que el angular de una cámara web no es muy grande y que no está preparada para tomar fotos con muy poca luminosidad, hubo un proceso de selección de la cámara para encontrar la más adecuada para el proyecto. Además, a esta cámara se le ha cambiado la lente a otra con una longitud focal inferior para conseguir una imagen lo más panorámica posible.

Para la selección de la cámara busqué en webs de astrofotografía, ya que se precisa una webcam con la mayor captación de luz posible. En estas páginas y foros hablan de unas cámaras con sensores CCD (normalmente las webcams llevan sensores CMOS), los cuales consiguen mejores resultados en lugares poco iluminados. El problema que encontré fue que toda la información era bastante antigua, y hablaban de modelos de webcams ya descatalogados (de 2005-2007 aproximadamente). Valoraban especialmente bien de una cámara, la Philips SPC900NC Pixel Plus, para la cual incluso había tutoriales para modificarla y conseguir mayores tiempos de exposición (http://www.astrosurf.com/goat/documentos/taller/spc900nc/index.html). También

encontré una interesante lista de webcams con sensores CCD, pero como ya he comentado antes, todas son antiguas y por tanto están descatalogadas (http://www.rkblog.rk.edu.pl/astro/kamery-ccd/).

Una vez descartadas estas opciones, busqué en el mercado actual webcams con estos sensores CCD, pero desgraciadamente ya no las hacen, por tanto me basé para escoger las posibles candidatas en la calidad de imagen que podían dar y en la facilidad de poder extraerle la lente y ponerle una de gran angular. Las elegidas fueron las siguientes:

- Microsoft Lifecam Cinema: Tiene un diseño muy cómodo para cambiar la lente y una calidad de imagen aceptable. En diferentes reviews hablan de que tiene un sistema de autorregulación del tiempo de exposición, lo cual hace que consiga muy buenas imágenes en condiciones de poca luz.

Ilustración 7 – Microsoft Lifecam Cinema

- Creative Live! Cam Connect HD: es la más cara de las tres, pero tiene una muy buena calidad de imagen.

Ilustración 8 – Creative Life! Cam Connect HD

- Logitech HD Webcam C525: de las tres escogidas me parece la más pobre y complicada de cambiarle la lente, pero Logitech es una marca de confianza. Viendo reviews de esta y de su modelo superior, este último presenta una calidad mucho mayor de imagen pero es demasiado caro.

Ilustración 9 – Logitech HD Webcam C525

Finalmente escogí la Microsoft por su calidad/precio, además del sistema de regulación del tiempo de exposición dependiendo de la luz entrante.

Una vez teniendo la cámara web, pensé en cómo debía hacer el tratamiento de la imagen para poder detectar las nubes en esta. El primer método que intenté fue por medio de algoritmos de segmentación por detección de objetos, ya que con ellos somos capaces de extraer de imágenes objetos precisos. El problema que conllevó este método fue que, dado que las nubes tienen formas amorfas, es imposible crear un patrón que las identifique como tal.

Vista de esta manera que no se podía hacer el análisis de la imagen por formas, traté de hacerlo por el color, ya que en una imagen al cielo, si analizamos sus píxeles, los que tengan valores de saturación menores serán aquellos que correspondan a las nubes por sus colores de la gama de blancos y grises. Hice con una cámara réflex fotos a diversos tipos de cielo para determinar el umbral de blancos saturados mínimo que debía tener la foto para que considerase que había nubes, ya que hay que tener en cuenta factores de alta luminosidad como el sol o la luna. Para ello, se desarrolló un pequeño programa que crea un diagrama de colores HSV de la imagen cargada en este.

El modelo HSV (del inglés Hue, Saturation, Value ‑ Matiz, Saturación, Valor), también llamado HSB (Hue, Saturation, Brightness ‑ Matiz, Saturación, Brillo), define un modelo de color en términos de sus componentes.

La ruleta de color HSV representa el matiz como una región circular; una región triangular separada, puede ser usada para representar la saturación y el valor del color. Normalmente, el eje horizontal del triángulo denota la saturación, mientras que el eje vertical corresponde al valor del color. De este modo, un color puede ser elegido al tomar primero el matiz de una región circular, y después seleccionar la saturación y el

valor del color deseados de la región triangular.

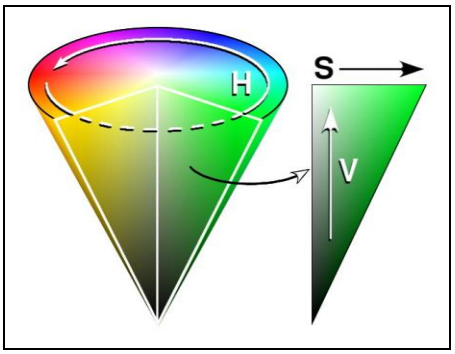
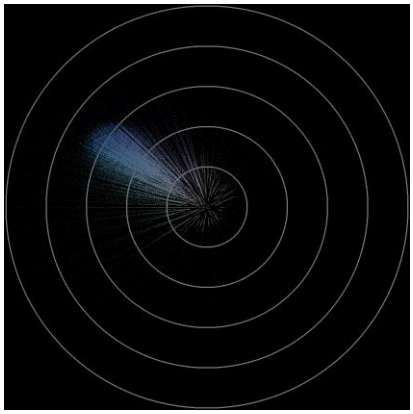
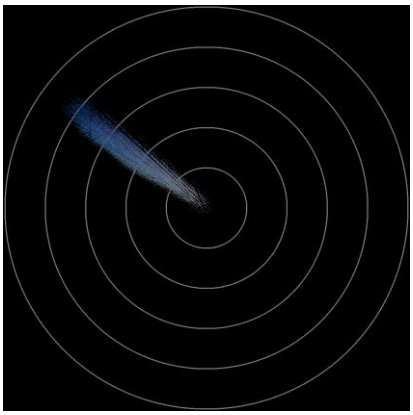
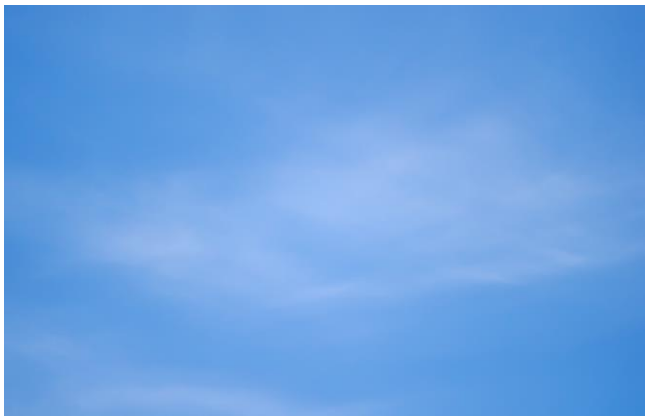
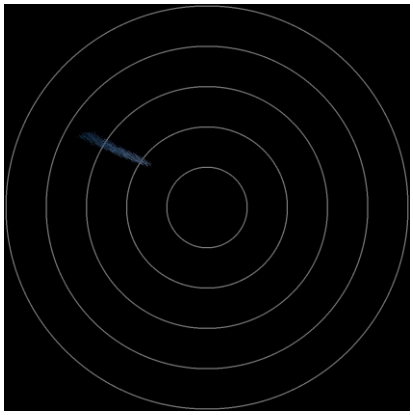
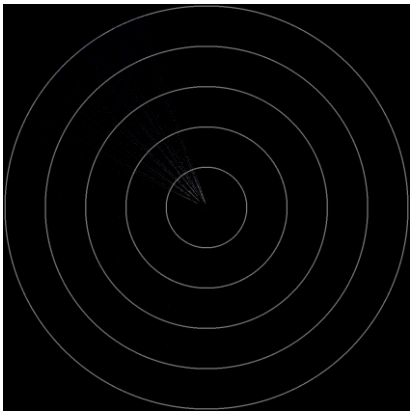
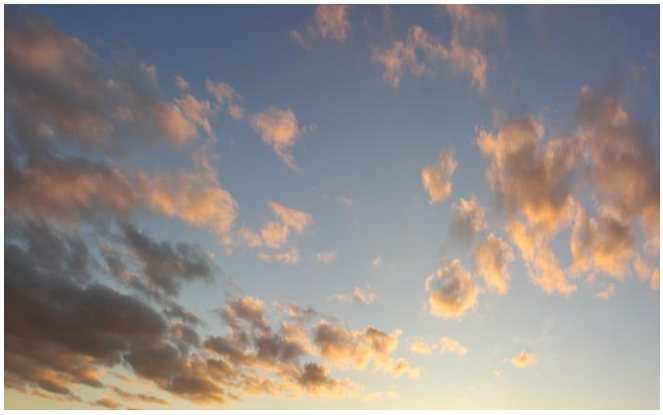
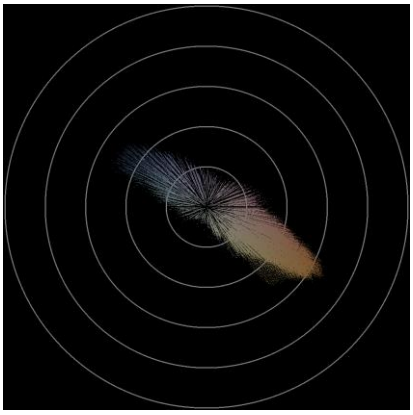
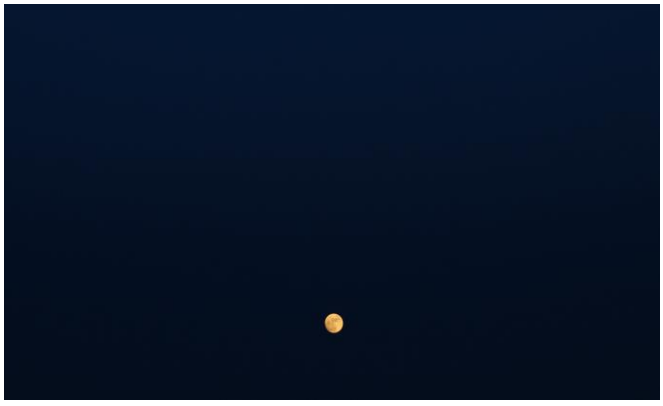
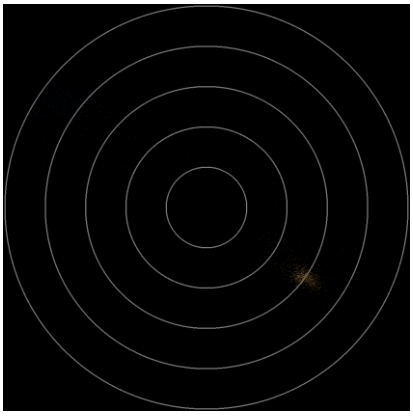
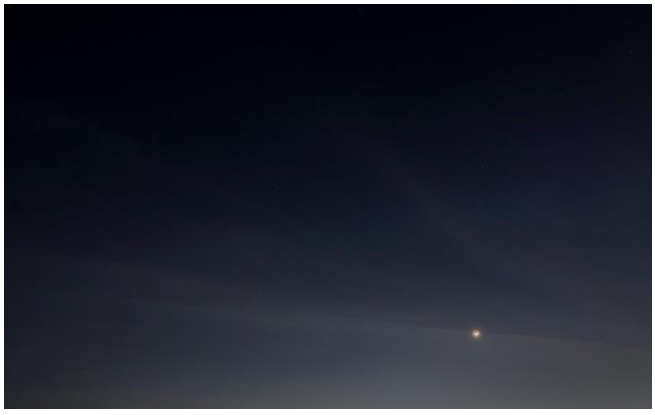
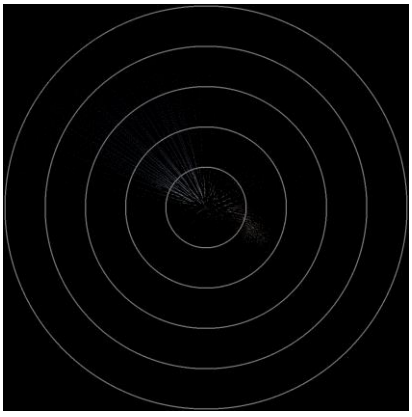


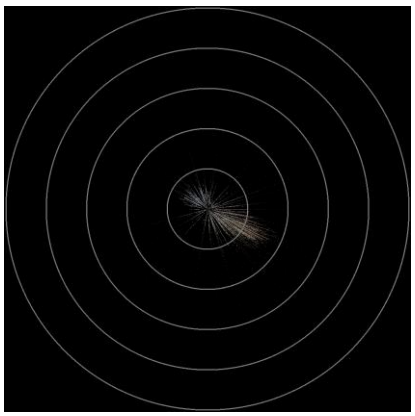
Ilustración 10 – Cono de colores del espacio HSV

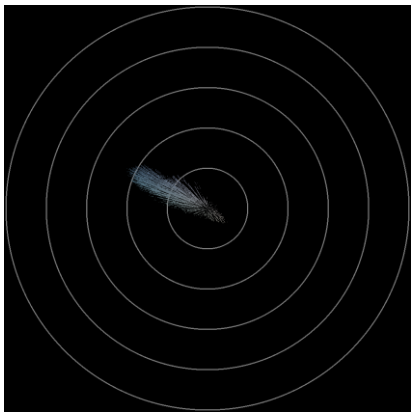
Algunos de los ejemplos con diferentes cielos son los siguientes:



Tras estudiar los resultados obtenidos con todas las imágenes creadas, llegamos a la conclusión de que el estudio del color y la saturación de la imagen no eran suficientes para determinar si en una imagen había o no nubes.

Debido a la lentitud en el procesado de datos de las imágenes y el consecuente calentamiento de la Raspberry Pi, se llegó a la conclusión de que era mejor mantener esta parte del proyecto fuera de la RPi, realizando los cálculos con un programa externo en el ordenador.

Finalmente, sólo esta parte del proyecto con el tratamiento de la imagen para detectar las nubes resultó requerir demasiado tiempo, por lo que decidimos dejar esta parte de la estación meteorológica para el futuro proyecto de fin de Máster (TFM), dejando sólo para el TFG la monitorización de la temperatura, humedad y velocidad del viento con la Raspberry Pi.

5.6. Comunicación de la Raspberry Pi con el ordenador

Hablando con el tutor, me comentó que la mejor manera de comunicar, en este caso, la Raspberry Pi con un ordenador era con un protocolo TCP/IP, el cual es sencillo de implementar y funciona por medio de internet con las IP's de los dispositivos.

Dado que Antonio ya tenía clientes y servidores TCP/IP funcionando en el software de los ordenadores para comunicarse entre sí, con el telescopio y con la cúpula, sólo tenía que crear el servidor en la RPi con todos los datos del programa unificado del anemómetro y del sensor de temperatura y humedad y desarrollar el protocolo de comunicación con los ordenadores.

Para comprobar el correcto funcionamiento del servidor sin tener que estar en el telescopio, hice un cliente telnet desde la misma Raspberry para conectarme con el servidor y corroborar que devolvía los datos pedidos por pantalla.

Finalmente, con el funcionamiento correcto del servidor, sólo quedaba hacerlo un archivo ejecutable de manera que cada vez que se encendiese la Raspberry Pi se ejecutase el programa automáticamente. Con esto hecho, el trabajo con la RPi está terminado.

6. Desarrollo

En este apartado de la memoria se explicará detalladamente todos los pasos realizados en el proyecto para conseguir el correcto funcionamiento de sus diferentes funciones.

6.1. Instalación del SO y puesta a punto de la Raspberry Pi

El primer paso necesario para empezar el proyecto es instalar el sistema operativo en la tarjeta Micro SD de nuestra Raspberry Pi modelo B+. El paquete que vamos a instalar en nuestra RPi es Raspbian, que utiliza como sistema operativo Debian Wheezy.

Debian, o Proyecto Debian, es una comunidad conformada por desarrolladores y usuarios, que mantiene un sistema operativo GNU basado en software libre. El sistema se encuentra precompilado, empaquetado, y en formato .deb para múltiples arquitecturas de computador y para varios núcleos. La primera adaptación del sistema Debian, siendo también la más dearrollada, es Debian GNU/Linux, basada en el núcleo Linux, y como siempre utilizando herramientas de GNU.

Una vez explicado brevemente el sistema operativo escogido para grabar en la tarjeta Micro SD, damos los pasos para la grabación, instalación y configuración de Raspbian.

Previamente necesitaremos un lector de tarjetas Micro SD con puerto USB para poder conectarlo al ordenador e instalarle Raspbian. Con nuestra tarjeta ya conectada entramos en https://www.raspberrypi.org/downloads/, donde podremos descargar en nuestro ordenador Raspbian.

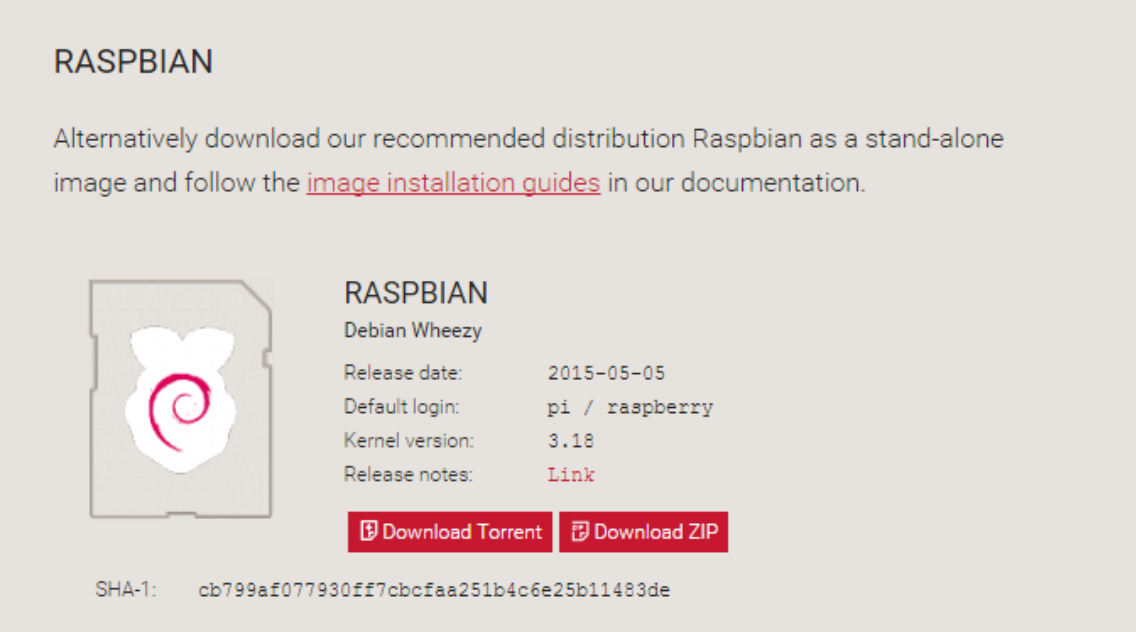


Ilustración 11 – Descarga del SO Raspbian

Una vez descargada la imagen del disco del SO necesitamos un programa de escritura de imágenes para poder instalarla en la Micro SD. Dado que yo tengo un Windows he usado el programa Win32DiskImager, el cual podemos descargar de http://sourceforge.net/projects/win32diskimager/.

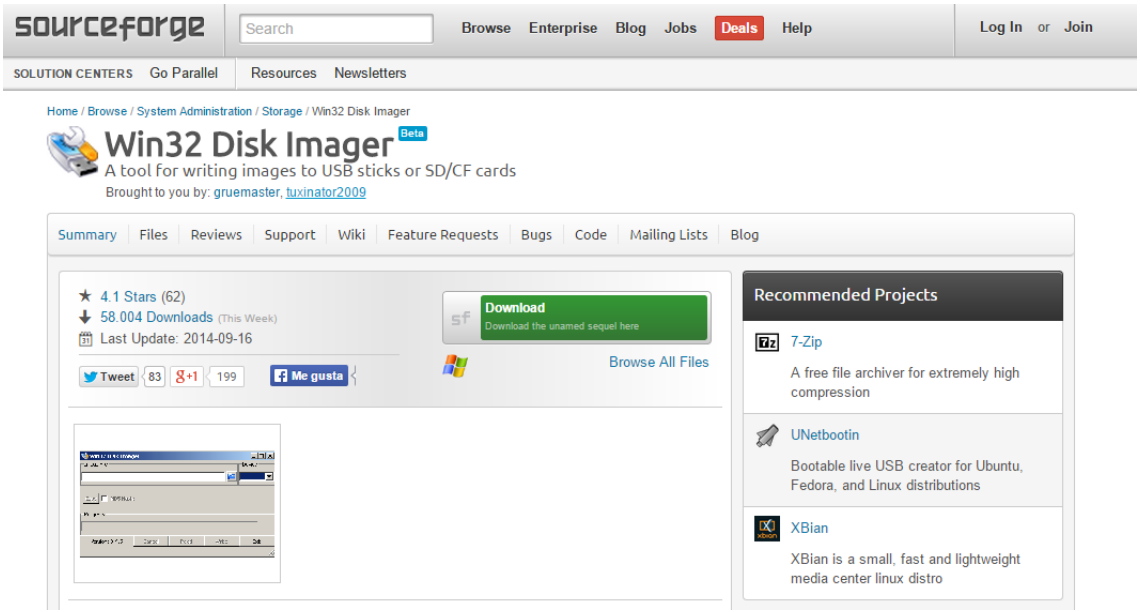


Ilustración 12 – Descarga de Win32 Disk Imager

Cuando se haya descargado, descomprimimos el archivo .zip y ejecutamos la utilidad Win32DiskImager. Es posible que sea necesario ejecutarla como administrador, para ello haz click en el botón derecho sobre la utilidad y selecciona “Ejecutar como administrador”.

Dentro del programa abrimos el archivo de imagen de Raspbian y seleccionamos como dispositivo a guardar la imagen la tarjeta micro SD (en mi caso es G:\); ahora sólo queda darle al botón "Write" para copiarlo.

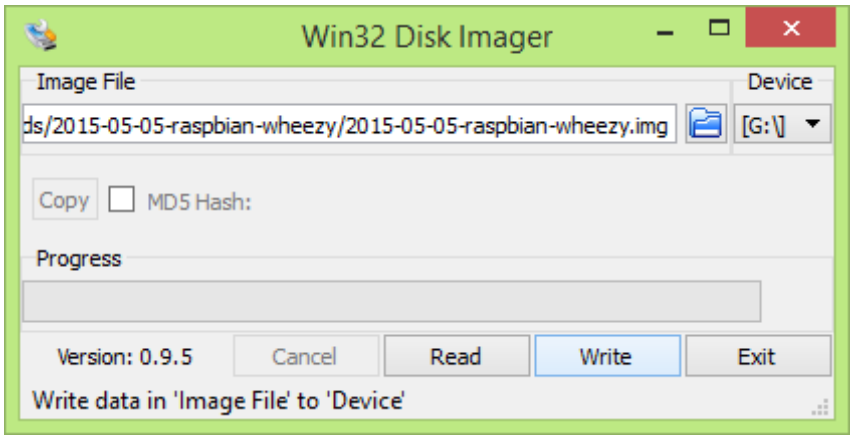


Ilustración 13 – Interfaz de Win32 Disk Imager

Con la tarjeta ya lista con el sistema operativo ya podemos insertarla en la Raspberry Pi y ejecutarla para configurar los primeros elementos. La primera vez que arrancamos la Raspberry Pi con Raspbian, debería iniciarse una pantalla azul tipo MSDOS, BIOS o Terminal como la siguiente.

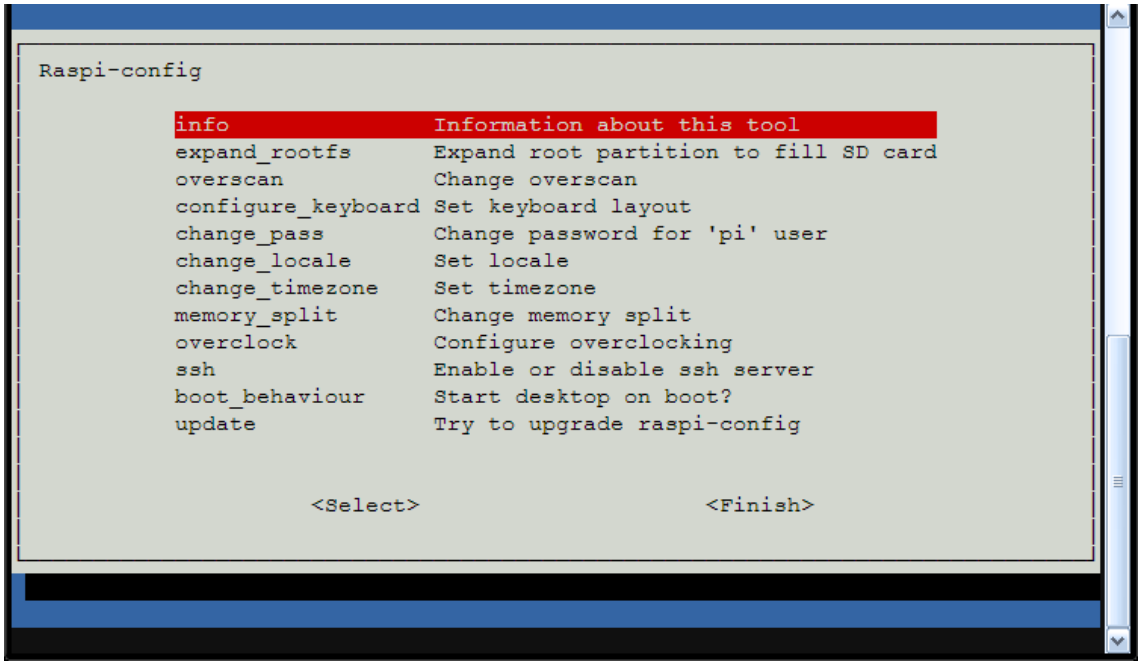


Ilustración 14 – Menú de configuración principal del Raspi-config

Si no se abre este menú o queremos volver a cambiar algo en la configuración podemos ejecutarlo tecleando el comando:

```
sudo raspi-config
```

Ahora vamos a ver las opciones una por una:

1. **“ info – Information about this tool”**

Información (en inglés) sobre esta herramienta de configuración.

2. **“expand_rootfs – Expand root partition to fill SD card”**

La tarjeta SD tendrá dos particiones: una de boot (arranque) que hace las veces de BIOS y que es visible poniendo la tarjeta en un PC con Windows o Mac y otra principal, donde está instalado Raspbian y que solo es visible en un PC con Linux. Como las imágenes por defecto que se usan como base para instalar Raspbian son de 2GB, si hemos usado una tarjeta micro SD más grande el resto del espacio estará sin utilizar, como es nuestro caso ya que hemos empleado una tarjeta micro SD de 16GB. Al usar esta opción del menú de configuración haremos que Raspbian aproveche todo el espacio restante de la tarjeta.

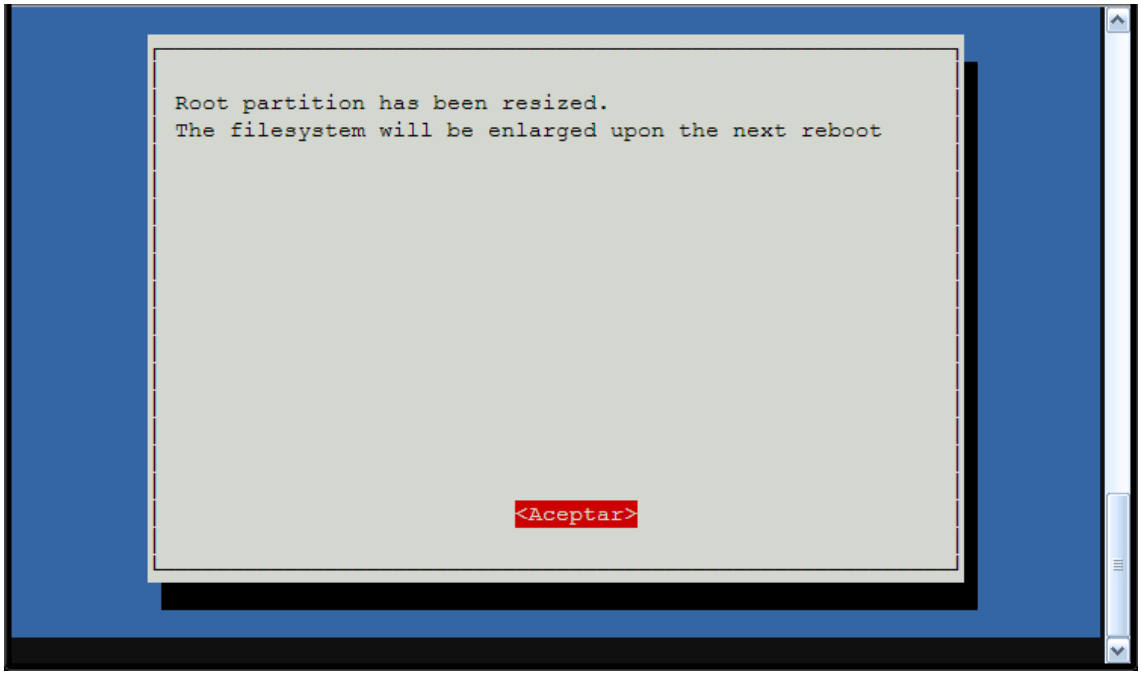


Ilustración 15 – Configuración de la partición de la tarjeta micro SD

3. “overscan – Change overscan”

Si la imagen que parece en pantalla no ocupa completamente el monitor podemos probar a desactivar esta opción y ver si se corrige.

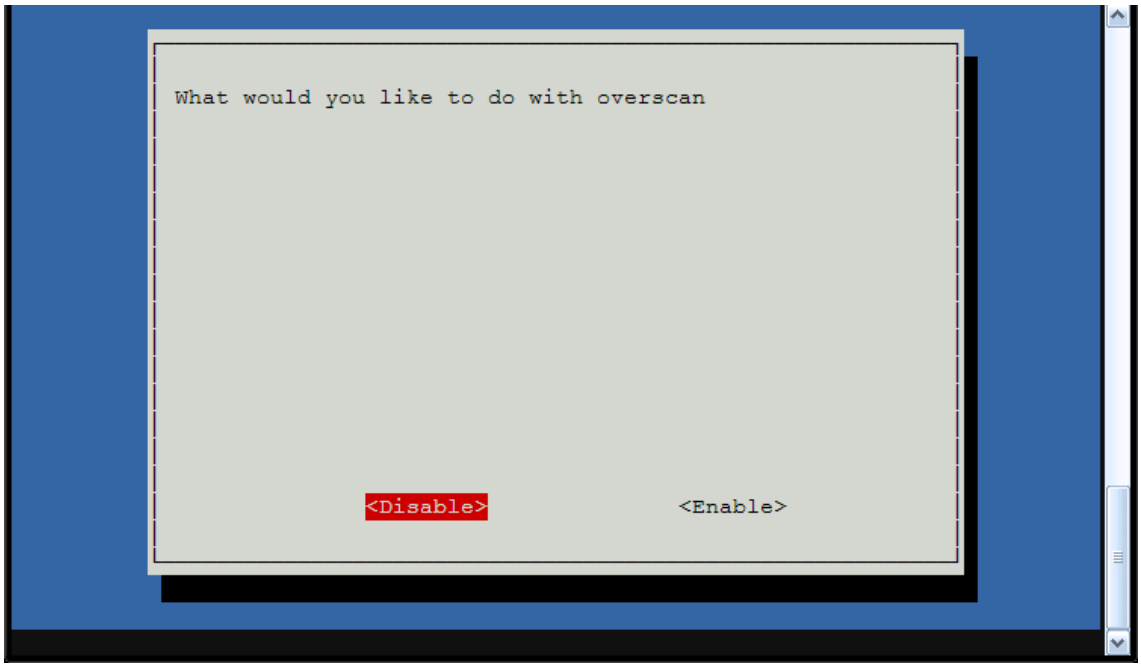


Ilustración 16 – Configuración del Overscan

4. “configure_keyboard – Set keyboard layout”

Para configurar el teclado hay que hacer varios pasos:

1) El tipo de teclado lo más normal es que sea “Generic 105-key (Intl) PC”.

2) Luego el idioma para usar con el teclado. Por defecto está configurado en inglés británico. Vamos a “Other” y nos movemos con los cursores del teclado (las flechas) hasta tener “Spanish”.

3) A continuación solicitará qué tipo de idioma es el teclado (además del que hemos dicho que íbamos a usar en el punto anterior). Puede ser que usemos un teclado español con Ñ (como en mi caso) o aunque usemos el idioma Spanish nuestro teclado sea English. Lo normal es un teclado también “Spanish”.

4) Ubicación de la tecla “Alt”: Lo normal es coger “The default for the

keyboard layout".

5) Luego pedirá la tecla para componer "AltGr" (compose key). Igualmente lo normal suele ser usar "No compose key".

6) Lo último que hay que indicar es si deseamos usar la combinación de teclas "Ctrl + Alt + Borrar" para salir del modo de interfaz gráfica. Por defecto, la opción es NO.

5. "change_pass – Change password for 'pi' user"

Para cambiar la contraseña de fábrica. Por defecto es: user 'pi' y pass 'raspberry'. Por seguridad es recomendable cambiarla, aunque yo no lo he hecho.

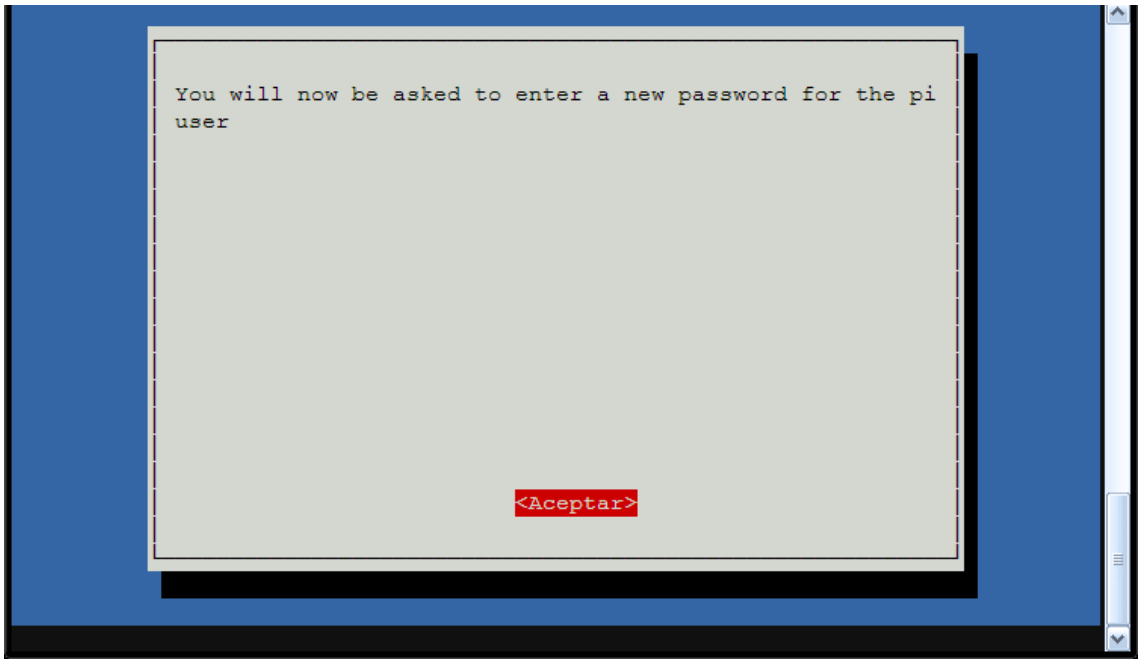


Ilustración 17 – Configuración de la contraseña de usuario de la RPi

6. "change_locale – Set locale"

Desde esta opción se configura el idioma para el sistema operativo. Además de que cambia la forma de presentar los números, la moneda y las fechas conforme a lo estándar en ese idioma, también cargará las ayudas y las aplicaciones en ese idioma siempre que exista traducción. En este menú ya estará marcado el "en_GB", que vamos a dejarlo y además añadir el español, el cual lo buscas moviéndote con los

cursores (las flechas del teclado) y cuando lo encuentres pulsas a la tecla “espacio”. Si eliges “es_ES” se refiere a español de. El mapa de códigos recomendado que es UTF-8. Una vez elegido todo pulsamos la tecla “Enter”. Y lo siguiente será elegir cuál de los instalados es el principal. En nuestro caso “es_ES”, lo seleccionamos y le damos a aceptar.

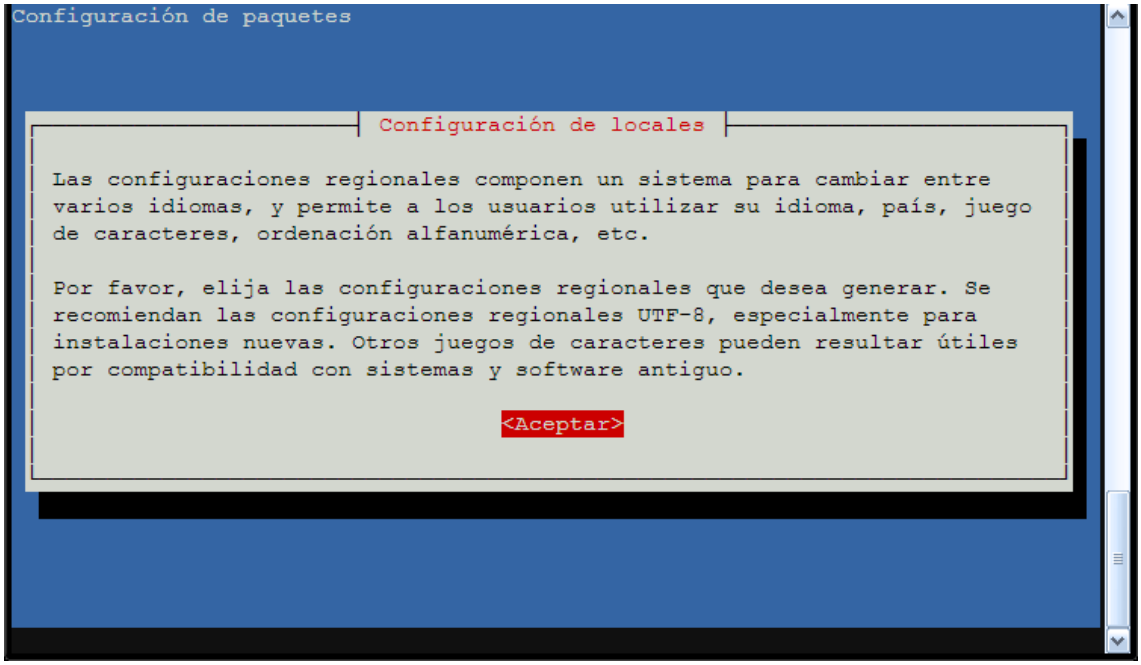


Ilustración 18 – Configuración del idioma en la RPi (I)

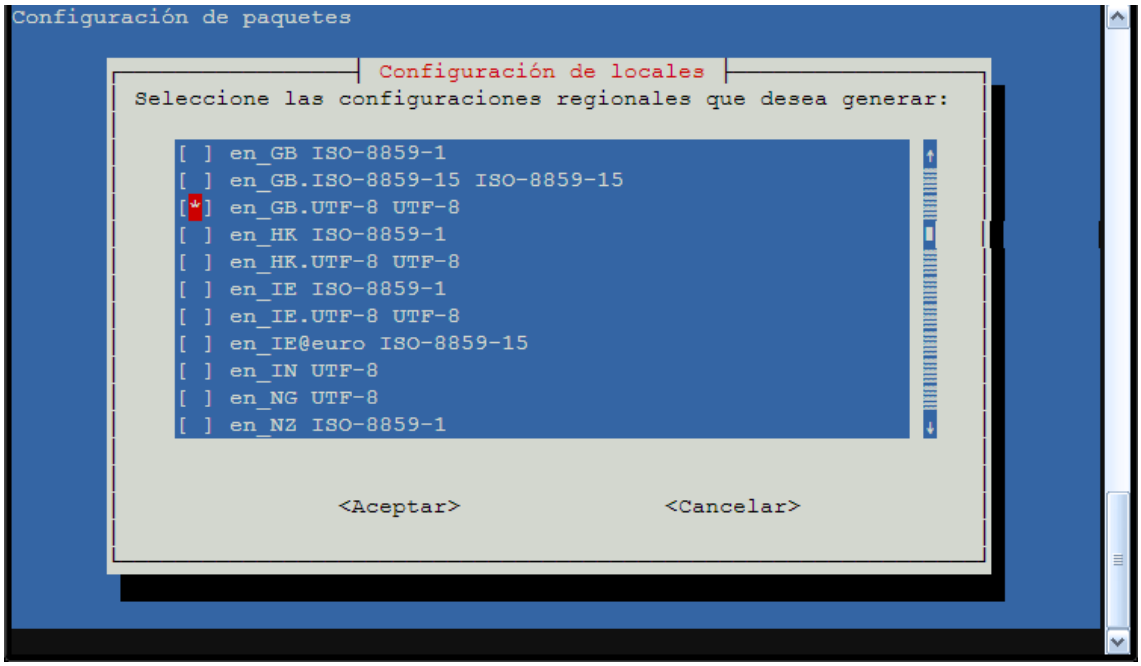


Ilustración 19 – Configuración del idioma en la RPi (II)

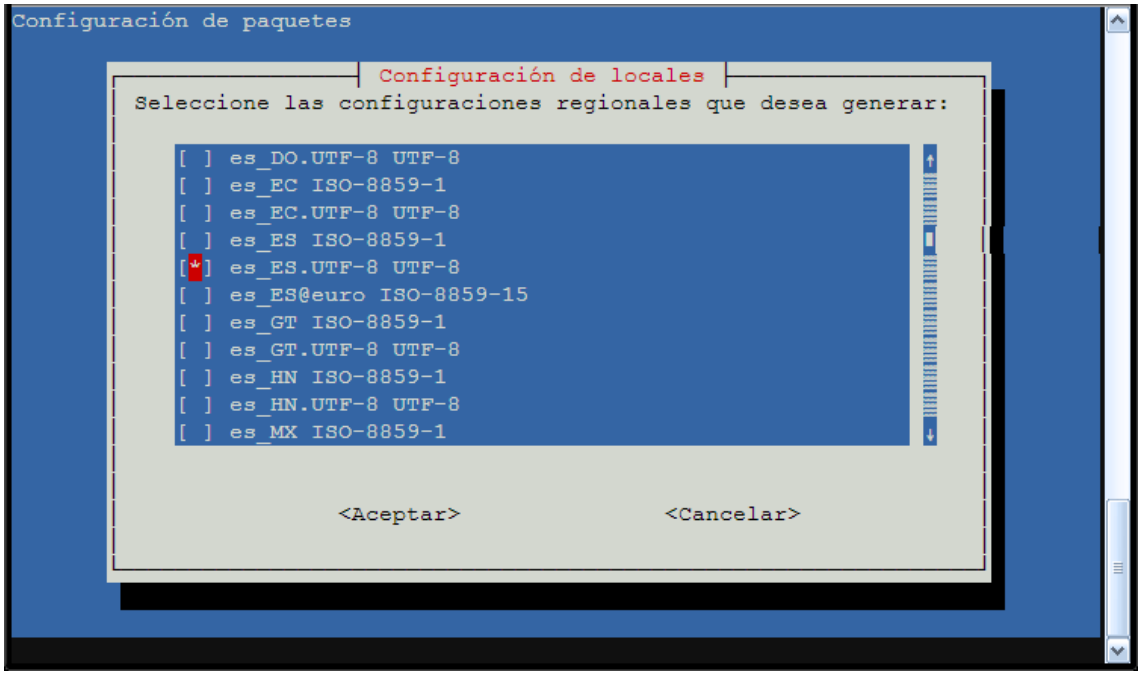


Ilustración 20 - Configuración del idioma en la RPi (III)

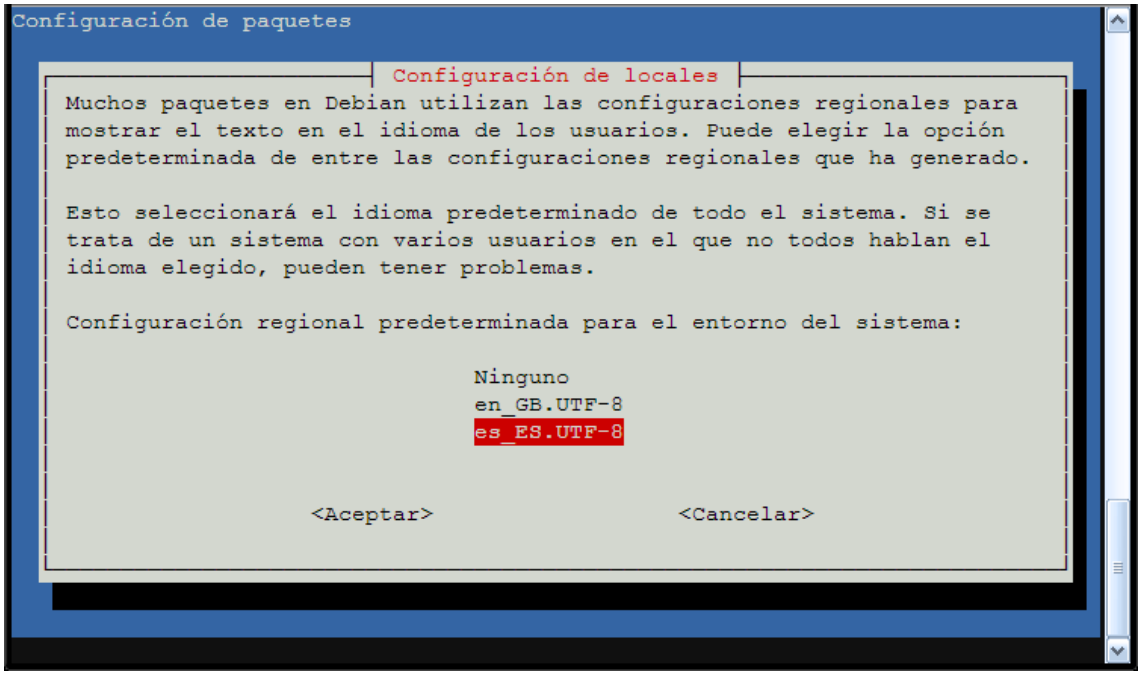


Ilustración 21 - Configuración del idioma en la RPi (IV)

7. "change timezone – Set timezone"

Esto es para elegir la hora de la Raspberry Pi. Primero la región continental "Europe" y luego la ciudad "Madrid. Hay que tener en cuenta que, como la Raspberry no tiene una batería ni un reloj interno tipo RTC, esta hora solo se usa como referencia para añadir o quitar horas sobre la hora estándar que la Raspberry lee de

Internet al conectarse, sin conexión no hay hora real.

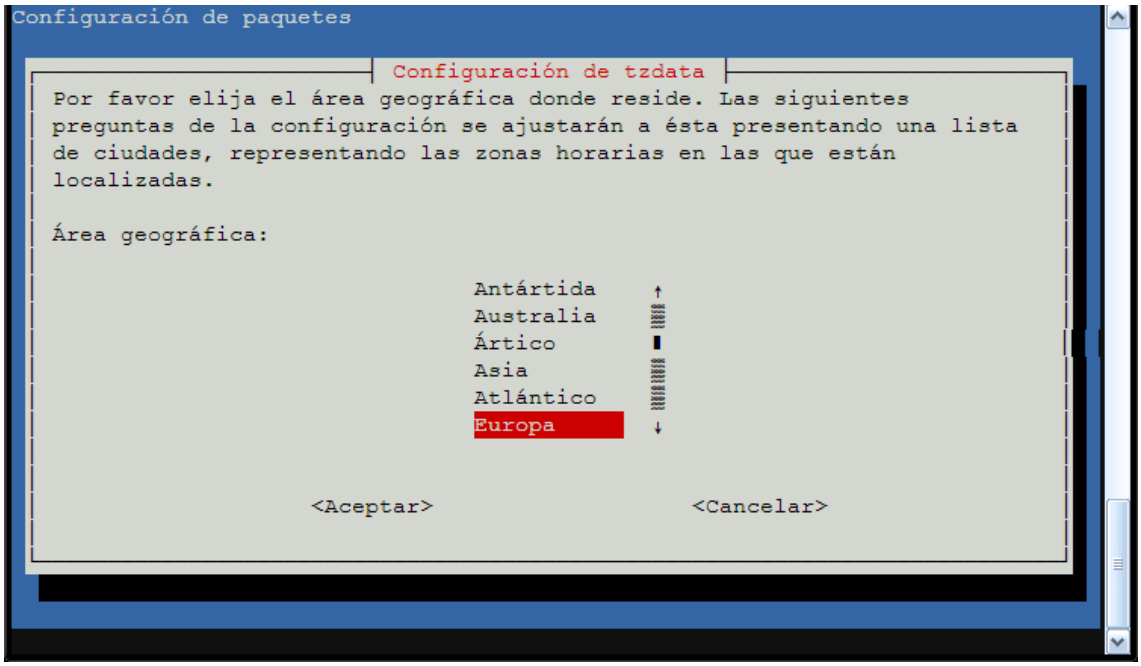


Ilustración 22 - Configuración de la zona horaria en la RPi (I)

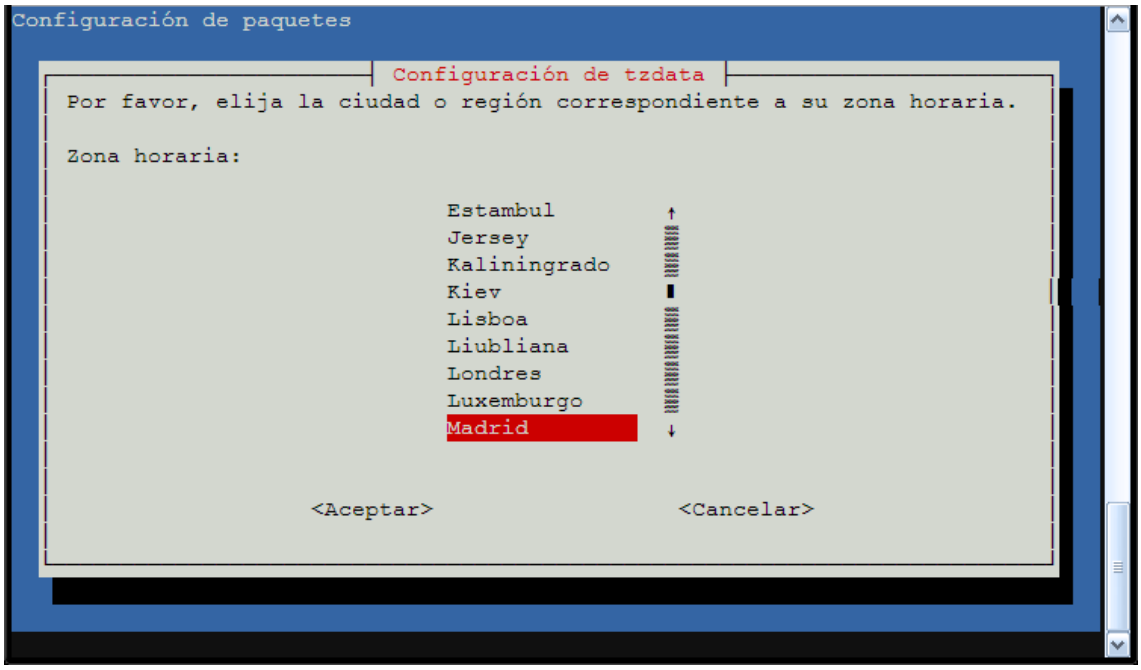


Ilustración 23 - Configuración de la zona horaria en la RPi (II)

8. "memory_split – Change memory Split"

En este apartado elegimos la cantidad de memoria que se destina a la tarjeta gráfica y cual para memoria principal. Para usarlo como servidor lo mejor es coger la mínima que sería 16. Si vamos a usarlo conectado a una pantalla como PC una solución

de compromiso para los modelos antiguos de 256MB será coger la de 64. Y si ya disponemos de una de 512MB puede ser mejor la de 128. Las versiones con XBMC ya cambian esto por su cuenta para optimizar, como la nuestra, por tanto no es necesario tocar nada.

9. "overclock – Configure overclocking"

Aquí podremos escoger entre varias configuraciones para forzar la velocidad del procesador, memoria, SD, etc. La capacidad de nuestra Raspberry cambiará mucho desde los 700MHz a los 1000MHz y es cuestión de suerte cómo sea de estable. Hay que tener en cuenta que no se pierde la garantía y no tiene por qué romperse la Raspberry por hacerlo, pero sí que puede reducir su vida a largo plazo, por tanto no vamos a tocarlo.

10. "ssh – Enable or disable ssh server"

En esta opción activamos el acceso remoto por terminal a la Raspberry y que por defecto está activo. Es mejor tenerlo así siempre, ya que es imprescindible si vamos a usar un monitor.

11. "boot_behaviour – Start desktop on boot?"

Al elegir "Yes", arrancará el escritorio tipo Windows directamente al encender la Raspberry. Si elegimos "No" arrancará en modo Terminal y luego de forma manual con el comando startx podemos arrancarlo nosotros cuando queramos. Si estamos conectados por Terminal Putty (es decir, conectados directamente a un PC), o el que sea, nunca veremos este entorno gráfico.

Ilustración 24 - Configuración del escritorio en la RPi

12. "update – Try to upgrade raspi-config"

En esta opción aplicamos todos los cambios realizados en raspi-config, actualizando todo el sistema operativo de Raspbian.

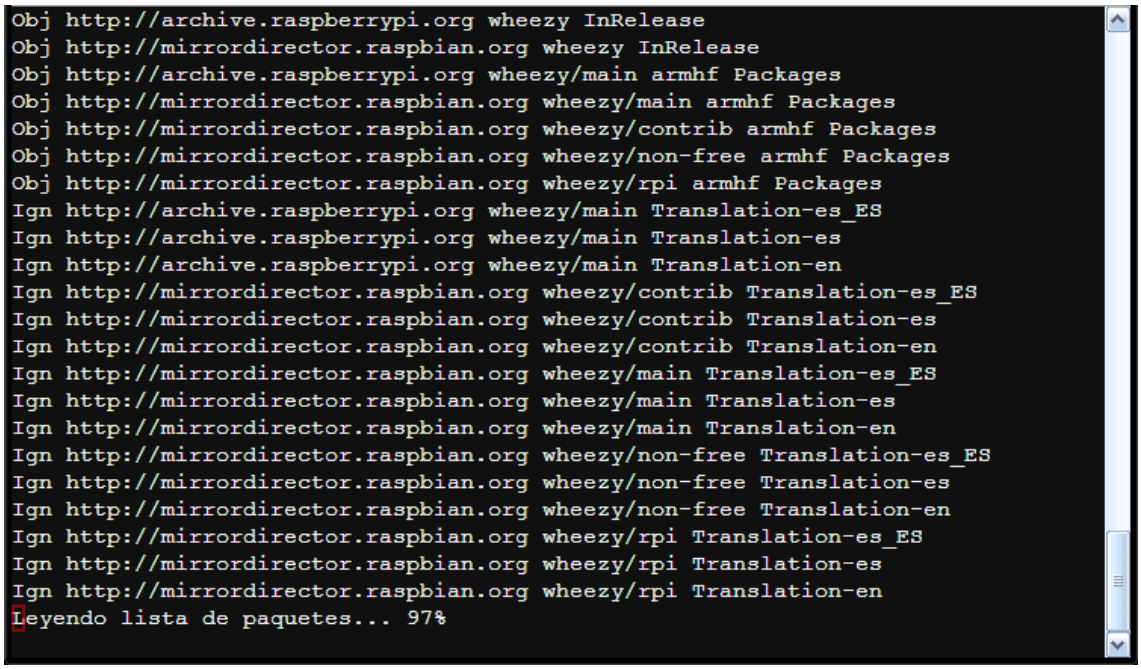


Ilustración 25 – Upgrade de Raspi-config

Luego nos vamos a "Finish". Termina la configuración de Raspbian y se guardan los cambios realizados. Ya solo queda reiniciar para confirmar que todo está OK.

Una vez reiniciada la Raspberry Pi accederemos directamente a la interfaz de escritorio.

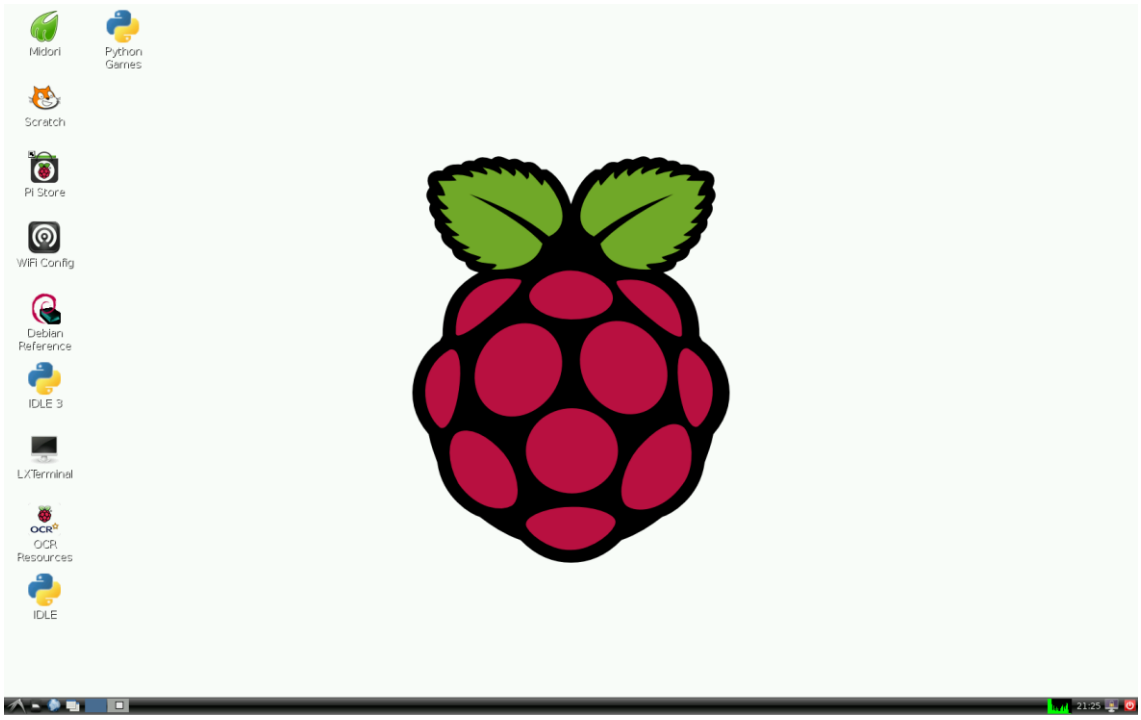


Ilustración 26 – Escritorio de la Raspberry Pi

El siguiente paso será configurar el WiFi de la Raspberry Pi; para ello será necesario tener conectado a esta un dispositivo inalámbrico de WiFi USB. La manera más simple de realizar esta tarea es clicando en el icono del escritorio "WiFi Config".

Ilustración 27 – Icono de WiFi Config del escritorio

Una vez dentro nos aparecerá una ventana como esta:

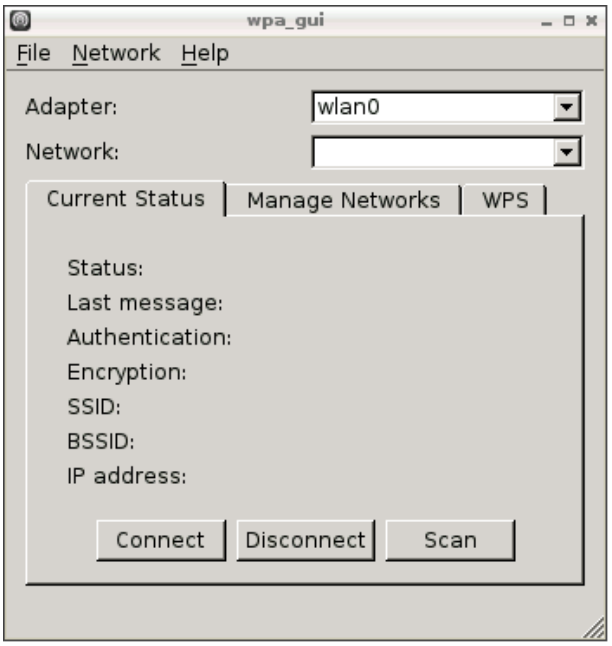


Ilustración 28 – Configuración de WiFi Config (I)

Pulsaremos al botón "Scan", donde nos abrirá otra ventana con todas las redes disponibles detectadas por nuestro dispositivo USB inalámbrico. Pulsamos en la que nos interese y se nos abrirá una ventana similar a la siguiente:

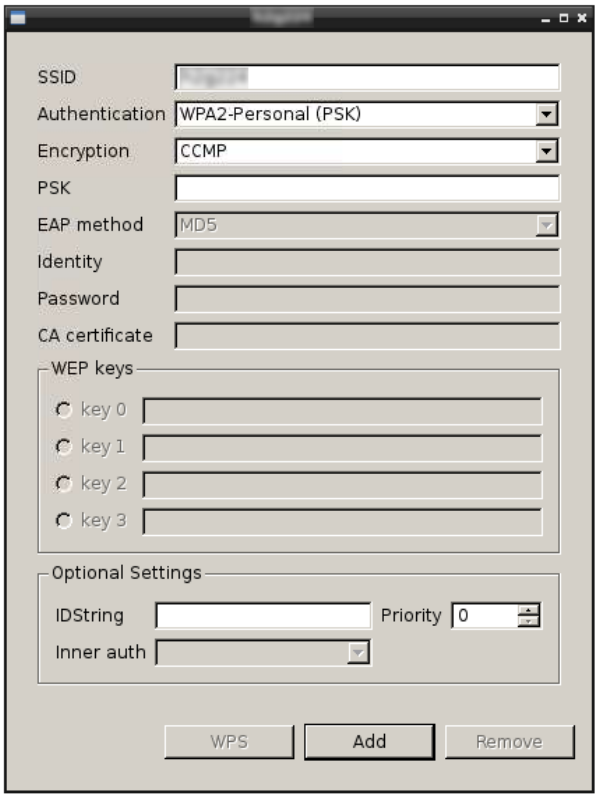


Ilustración 29 – Configuración de WiFi Config (II)

Dentro del apartado “PSK” escribiremos la contraseña de nuestra red, y después de esto sólo tendremos que darle al botón “Add”. Si la contraseña se ha escrito correctamente nos conectará automáticamente a la red seleccionada.

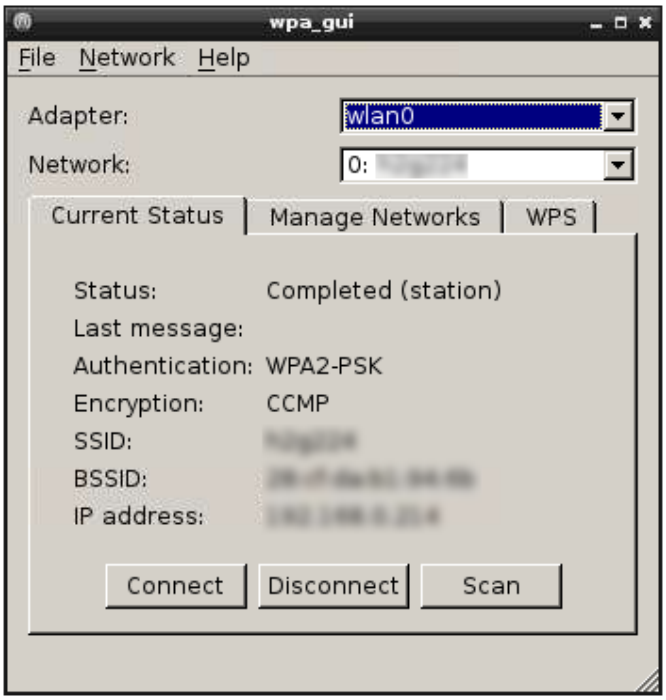


Ilustración 30 – Configuración de WiFi Config (III)

El siguiente y último paso a realizar en este apartado es instalar varias librerías necesarias para empezar a programar en Python nuestros programas mediante la ventana de comandos LXTerminal. La versión de Python que vamos a usar es la 2.7.

Primeramente ejecutaremos los comandos que actualizan los paquetes existentes o instalan algunos nuevos que el sistema operativo considera imprescindibles:

```
sudo apt-get update
sudo apt-get upgrade
```

Tras estos comandos instalaremos los paquetes necesarios para Python 2.7:

```
sudo apt-get install python
sudo apt-get install python-dev
sudo apt-get install libjpeg-dev
sudo apt-get install python-rpi.gpio
sudo apt-get install python-pip
sudo apt-get install python-setuptools
```

Instalaremos también los paquetes relacionados con el Bus GPIO de la Raspberry, entre otros:

```
sudo pip install RPi.GPIO
sudo pip install pySerial
sudo pip install nose
```

Otro conjunto de paquetes importantes es el de operaciones matemáticas y representaciones gráficas en 2D:

```
sudo apt-get install python-matplotlib
sudo apt-get install python-mpltoolkits.basemap
sudo apt-get install python-numpy
sudo apt-get install python-scipy
```

Finalmente, instalaremos el paquete de GIT, el cual es útil para descargar archivos de internet poniendo la dirección de la página.

```
sudo apt-get install git
```

6.2. Sistema de monitorización de la velocidad del viento

Primeramente, para poder conectar el anemómetro de salida analógica necesitamos un conversor analógico/digital, que en nuestro caso será el MCP3008.

La Raspberry Pi no tiene manera de leer las entradas analógicas. Es un dispositivo completamente digital, comparándolo con el Arduino, AVR o microcontroladores PIC que a menudo tienen 6 o más entradas analógicas. Las entradas analógicas son muy útiles porque muchos sensores tienen salidas analógicas, por lo que necesitan una manera de hacer que la RPi pueda manejarse con ellos.

Cablearemos el chip MCP3008 a ella. El MCP3008 actúa como un "puente" entre el modo digital y analógico. Dispone de 8 entradas analógicas que la RPi puede consultar usando 4 pines digitales. Mostramos a continuación un diagrama con los nombres de sus pines para poder cablearlos correctamente:

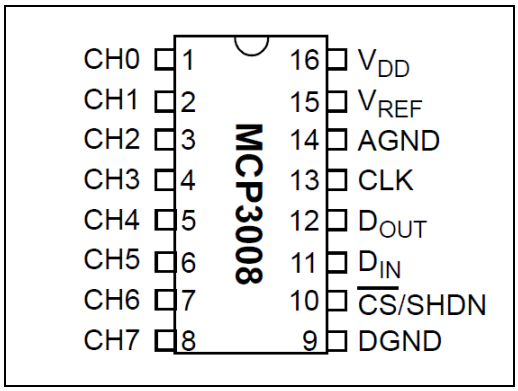


Ilustración 31 – Esquema del Datasheet de conversor A/D MCP3008

Este conversor tiene 7 entradas analógicas libres lo que permite una fácil ampliación del número de sensores (sensor de presión).

Para leer los datos analógicos necesitamos los siguientes pines del chip MCP3008 conectados a los pines de la Raspberry Pi:

Pines del MCP3008	**Pines de la RPi**	**Color del cable**
VDD (Alimentación)	3.3V	
VREF (Voltaje analógico de referencia	3.3V	
AGND (Tierra analógica)	GND	
CLK (Reloj)	#18	
DOUT (Datos de salida del MCP3008)	#23	
DIN (Datos de entrada de RPi)	#24	
CS (Selección de chip)	#25	
DGND (Tierra digital)	GND	

Tras esto conectaremos el pin #1 del MCP3008 (CH0) a la salida de voltaje analógica del anemómetro (cable azul) y la tierra del anemómetro a GND (cable negro). La problemática que surge con este sensor es que necesita una alimentación de 9V, la cual no puede suministrar la RPi ya que su máximo es 5V, por tanto es necesario conectar el cable de alimentación del anemómetro (cable marrón) a una alimentación externa de 9V fijos, conectada al suministro eléctrico. Uniremos el cable positivo de la fuente con el de alimentación del sensor y el negativo lo conectaremos a GND.

Haremos un esquema eléctrico para detallar las conexiones:

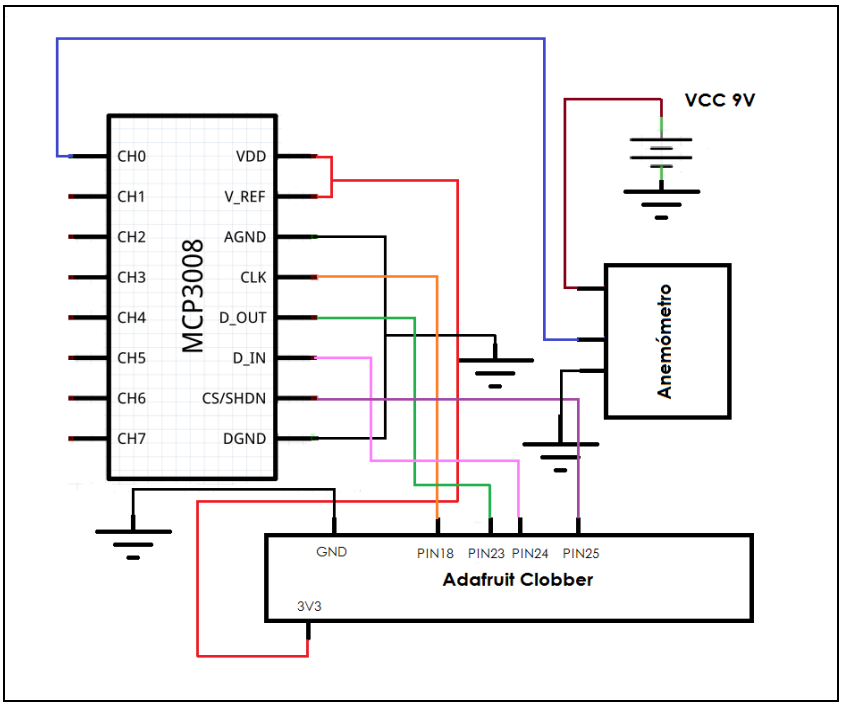


Ilustración 32 – Esquema eléctrico del anemómetro conectado a la RPi

Con este esquema gráfico podemos ver más claramente cómo se han realizado las conexiones:

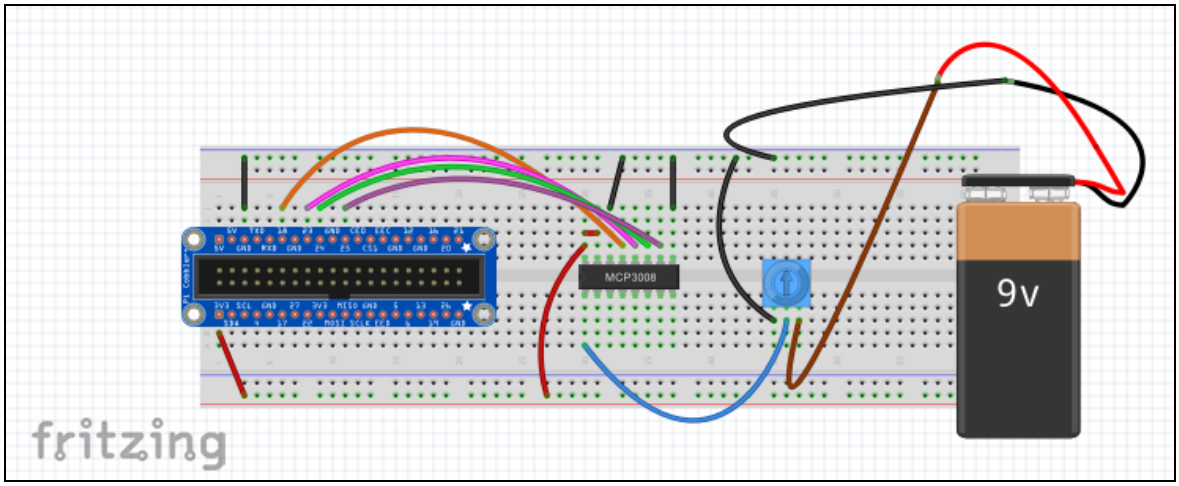


Ilustración 33 – Esquema gráfico del anemómetro conectado a la RPi

(*) Dado que no había una miniatura de un anemómetro, he usado para representarlo un pequeño potenciómetro azul.

Una vez cableado correctamente el MCP3008 y el anemómetro podemos crear el script de Python que los hará funcionar. El objetivo de nuestro programa es que lea los datos de salida (voltaje) del anemómetro nuestro conversor A/D, que en función del valor de estos de un valor digital, y que finalmente represente por pantalla el valor de la velocidad del viento en un bucle que se repite cada segundo. También medirá las

velocidades de ráfaga y el valor promedio. Además, si alguno de estos datos supera un umbral estipulado, mandará un mensaje de advertencia.

Como datos a conocer a la hora de convertir los valores de voltaje en valores de viento, se debe saber que el rango de valores de voltaje que nos da el anemómetro es desde 0.4V (0 m/s) hasta 2.0V (32.4 m/s), por tanto todos los valores inferiores a 0.4V corresponderán a 0 m/s; es posible que llegase a dar un valor superior a 2.0V, lo que implicaría una mayor velocidad del viento. El MCP3008 es un conversor A/D de 10 bits, lo cual implica que puede leer valores desde 0 hasta 1023 (2^{10} = 1024 valores), donde 0 equivale a "tierra" y 1023 equivale a 3.3V. Con estos datos podemos establecer una relación entre el valor de registro del conversor, el voltaje de salida del anemómetro y la velocidad del viento medida:

- Si dividimos 3.3V entre 1024 valores tendremos 0.0032226V por cada valor, por lo que 0.4V corresponde al registro 124, aproximadamente, y 2V corresponde a 621. Por tanto, desde el registro 0 hasta el 124 el valor de la velocidad del viento será 0.0 m/s y, a partir de 125 hasta 621 tendrá un valor que irá aumentando hasta unos 32 m/s.

- Si creamos una fórmula para calcular la velocidad del viento en función del voltaje de salida, suponiendo que 0.4V equivale a 0 m/s y 2V equivale a 32 m/s, conseguimos un ratio lineal de 2 m/s por cada 100mV, por tanto la ecuación es:

$$WindSpeed = (SensorVoltage - 0.4) * 10 * 2$$

Una vez tenemos hecha esta relación, podemos proceder a escribir el programa que nos dará por pantalla el valor de la velocidad del viento cada 20 segundos:

```
#!/usr/bin/env python
import time
import os
import RPi.GPIO as GPIO
import sys
GPIO.setmode(GPIO.BCM)
DEBUG = 1

# Lee los datos SPI del chip MCP3008, 8 posibles adc's (0 hasta 7)
def readadc(adcnum, clockpin, mosipin, misopin, cspin):
        if ((adcnum > 7) or (adcnum < 0)):
                return -1
```

```
        GPIO.output(cspin, True)
        GPIO.output(clockpin, False)  # empieza el clck en baja
        GPIO.output(cspin, False)     # empieza el  CS en baja

        commandout = adcnum
        commandout |= 0x18  # bit de inicio + bit unico de fin
        commandout <<= 3    # solo necesitamos enviar 5 bits

        for i in range(5):
                if (commandout & 0x80):
                        GPIO.output(mosipin, True)
                else:
                        GPIO.output(mosipin, False)
                commandout <<= 1
                GPIO.output(clockpin, True)
                GPIO.output(clockpin, False)

         adcout = 0

    # lee un bit vacio, un bit null y 10 ADC bits
        for i in range(12):
                GPIO.output(clockpin, True)
                GPIO.output(clockpin, False)
                adcout <<= 1
                if (GPIO.input(misopin)):
                        adcout |= 0x1

        GPIO.output(cspin, True)

        adcout >>= 1       # el primer bit es 'null', lo eliminamos
        return adcout

# Function para convertir los datos en nivel de voltaje,
# redondeado al numero de decimales que espedificaquemos (places).
def ConvertVolts(adcout,places):
        volts = (adcout * 3.3) / float(1023)
        volts = round(volts,places)
        return volts

# Funcion para calcular el viento de los datos del Anemometro
# redondeado al numero de decimales que espedificaquemos (places).
def ConvertWind(volts,places):

# Rango de voltaje del anemometro (0.4V-2.0V)
# cualquier valor debajo de 0.4V se considerara como 0 m/s
# Si evaluas 0.4 como 0 y 2.0 como 32 el ratio es linear a 2 m/s
por cada 100mV
  if volts <= 0.4:
        wind = 0
  else:
        wind = (volts - 0.4) * 10 * 2
```

```
  wind = round(wind,places)
  return wind

# Pines conectados desde el puerto SPI en el ADC hasta el Cobbler
SPICLK = 18
SPIMISO = 23
SPIMOSI = 24
SPICS = 25

# Configuramos los pines  de la interfaz SPI
GPIO.setup(SPIMOSI, GPIO.OUT)
GPIO.setup(SPIMISO, GPIO.IN)
GPIO.setup(SPICLK, GPIO.OUT)
GPIO.setup(SPICS, GPIO.OUT)

# La salida del anemometro OUTPUT conectada al ADC #0
wind_channel = 0;

while True:

        # Lee el pin analogico
        wind_level = readadc(wind_channel, SPICLK, SPIMOSI, SPIMISO,
SPICS)
        wind_volts = ConvertVolts(wind_level,2)
        wind = ConvertWind(wind_volts,2)

        if DEBUG:
            print "--------------------------------------------"
            print("Wind : {} ({}V) {}
m/s".format(wind_level,wind_volts,wind))

        # Esperamos 20 segundos a que reinicie el bucle
        time.sleep(20)
```

Ahora que hemos conseguido extraer los datos del anemómetro, le aplicaremos una alarma de viento, de manera que si supera la velocidad del viento de alarma estipulada mandará un mensaje de advertencia para que no se abra la cúpula. Como valor inicial estableceremos la velocidad de alarma en 8 m/s, con la posibilidad de cambiar este valor por pantalla al inicializar el programa.

```
WindWarning = 8 # Valor por defecto
WindWarning = sys.argv[1] # Lee el primer argumento tras el script
filename.py
print "Velocidad del viento de alarma marcada en: " + WindWarning +
" m/s"

while True:
        if DEBUG:
            if wind >= WindWarning:
                    print "Alerta! Cerrar cupula por viento"
```

Como elemento final añadido, calcularemos también la velocidad de ráfaga y la velocidad del viento promedio. Para conseguir estos valores, hemos visto en la web de AEMet que para realizar el cálculo de la velocidad del viento promedio se debe hacer una media con los valores tomados durante los últimos 10 minutos, y para las ráfagas se debe tomar el valor máximo de los valores medidos en 1 hora. La manera de realizar estos cálculos es aplicándole dos contadores a nuestro programa (inicializados en 0), que midan la velocidad del viento en vez de cada 20 segundos, cada 5, y que vayan sumando los valores de las medidas y aumentando el contador hasta llegar al valor deseado:

- Para el contador promedio: (10 minutos * 60 segundos) / 5 segundos = 120.

- Para el contador de ráfaga: (60 minutos * 60 segundos) / 5 segundos = 720.

Una vez alcanzado el valor máximo de cada contador, estos se inicializarían y el promedio realizaría la media dividiendo todos los valores sumados entre 120 y el de ráfaga tomaría el valor máximo de los medidos en dicha hora. Para este caso también pondremos que se pueda variar el valor de los contadores por pantalla y una señal de alarma para la velocidad de ráfaga:

```
count1 = 0 # inicializo contador 1 (vel. promedio)
count1_trigg = 600 # 10 min * 60 seg = 600 seg
count2 = 0 # inicializo contador 2 (vel. rafaga)
count2_trigg = 3600 # 1h * 60 min * 60seg = 3600 seg
windsum = 0 # inicializo la variable sumatorio
windraf = 0 # inicializo la variable de ráfaga

while True:

        # Lee el pin analogico
        wind_level = readadc(wind_channel, SPICLK, SPIMOSI,
SPIMISO, SPICS)
        wind_volts = ConvertVolts(wind_level,2)
        wind = ConvertWind(wind_volts,2)
        windsum = windsum + wind

        if float(wind) > float(windraf):
                windraf = wind

        windprom = windsum / ((count1 + 5) / 5)

        if DEBUG:
            print "--------------------------------------------"
            print("Tiempo calculando promedio: " + str(count1) + "
-> Vel. promedio: " + str(windprom) + "m/s")
```

```
            print("Tiempo calculando rafaga: " + str(count2) + " ->
Vel. rafaga: " + str(windraf) + "m/s")

        count1 = count1 + 5
        count2 = count2 + 5

        if int(count1) == int(count1_trigg):
                count1 = 0 # vuelvo a inicializar el contador
                windsum = 0 # inicializo sumatorio


        if int(count2) == int(count2_trigg):
                count2 = 0 # vuelvo a inicializar el contador
                windraf = 0 # inicializo el valor de rafaga

        # Esperamos 5 segundos a que reinicie el bucle
        time.sleep(5)
```

El código completo quedaría de la siguiente manera:

```
#!/usr/bin/env python

import time
import os
import RPi.GPIO as GPIO
import sys

GPIO.setmode(GPIO.BCM)
DEBUG = 1
WindWarning = 8 # Valor por defecto
WindWarning = sys.argv[1] # Lee el primer argumento tras el script
filename.py
print "Velocidad del viento de alarma marcada en: " + WindWarning +
" m/s"


# Lee los datos SPI del chip MCP3008, 8 posibles adc's (0 hasta 7)
def readadc(adcnum, clockpin, mosipin, misopin, cspin):
        if ((adcnum > 7) or (adcnum < 0)):
                return -1
        GPIO.output(cspin, True)

        GPIO.output(clockpin, False)  # empieza el clck en baja
        GPIO.output(cspin, False)     # empieza el  CS en baja

        commandout = adcnum
        commandout |= 0x18  # bit de inicio + bit unico de fin
        commandout <<= 3    # solo necesitamos enviar 5 bits

        for i in range(5):
                if (commandout & 0x80):
                        GPIO.output(mosipin, True)
```

```
                else:
                        GPIO.output(mosipin, False)

                commandout <<= 1
                GPIO.output(clockpin, True)
                GPIO.output(clockpin, False)

        adcout = 0

    # lee un bit vacio, un bit null y 10 ADC bits
        for i in range(12):
                GPIO.output(clockpin, True)
                GPIO.output(clockpin, False)
                adcout <<= 1
                if (GPIO.input(misopin)):
                        adcout |= 0x1

        GPIO.output(cspin, True)

        adcout >>= 1       # el primer bit es 'null', lo eliminamos
        return adcout


# Funcion para convertir los datos en nivel de voltaje,
# redondeado al numero de decimales que especifiquemos (places).
def ConvertVolts(adcout,places):
        volts = (adcout * 3.3) / float(1023)
        volts = round(volts,places)
        return volts

# Funcion para calcular el viento de los datos del Anemometro
# redondeado al numero de decimales que especifiquemos (places).
def ConvertWind(volts,places):

# Rango de voltaje del anemometro (0.4V-2.0V)
# cualquier valor debajo de 0.4V se considerara como 0 m/s
# Si evaluas 0.4 como 0 y 2.0 como 32 el ratio es linear a 2 m/s por
cada 100mV
  if volts <= 0.4:
        wind = 0

  else:
        wind = (volts - 0.4) * 10 * 2

  wind = round(wind,places)
  return wind

# Pines conectados desde el puerto SPI en el ADC hasta el Cobbler
SPICLK = 18
SPIMISO = 23
SPIMOSI = 24
SPICS = 25
```

```
# Configuramos los pines  de la interfaz SPI
GPIO.setup(SPIMOSI, GPIO.OUT)
GPIO.setup(SPIMISO, GPIO.IN)
GPIO.setup(SPICLK, GPIO.OUT)
GPIO.setup(SPICS, GPIO.OUT)

# La salida del anemometro OUTPUT conectada al ADC #0
wind_channel = 0;

count1 = 0 # inicializo contador 1 (vel. promedio)
count1_trigg = 600 # 10 min * 60 seg = 600 seg
count2 = 0 # inicializo contador 2 (vel. rafaga)
count2_trigg = 3600 # 1h * 60 min * 60seg = 3600 seg
windsum = 0 # inicializo la variable sumatorio
windraf = 0 # inicializo la variable de ráfaga

while True:
        # Lee el pin analogico
        wind_level = readadc(wind_channel, SPICLK, SPIMOSI, SPIMISO,
SPICS)
        wind_volts = ConvertVolts(wind_level,2)
        wind = ConvertWind(wind_volts,2)
        windsum = windsum + wind

        if float(wind) > float(windraf):
                windraf = wind

        windprom = windsum / ((count1 + 5) / 5)

        if DEBUG:
            print "--------------------------------------------"
           print("Wind : {} ({}V) {}
      m/s".format(wind_level,wind_volts,wind))
            print("Tiempo calculando promedio: " + str(count1) + " ->
Vel. promedio: " + str(windprom) + "m/s")
            print("Tiempo calculando rafaga: " + str(count2) + " ->
Vel. rafaga: " + str(windraf) + "m/s")
           if wind >= WindWarning:
                   print "Alerta! Cerrar cupula por viento"

        count1 = count1 + 5
        count2 = count2 + 5

        if int(count1) == int(count1_trigg):
                count1 = 0 # vuelvo a inicializar el contador
                windsum = 0 # inicializo sumatorio

        if int(count2) == int(count2_trigg):
                count2 = 0 # vuelvo a inicializar el contador
                windraf = 0 # inicializo el valor de rafaga

        # Esperamos 5 segundos a que reinicie el bucle
        time.sleep(5)
```

Finalmente, haremos una demostración del funcionamiento de nuestro código:

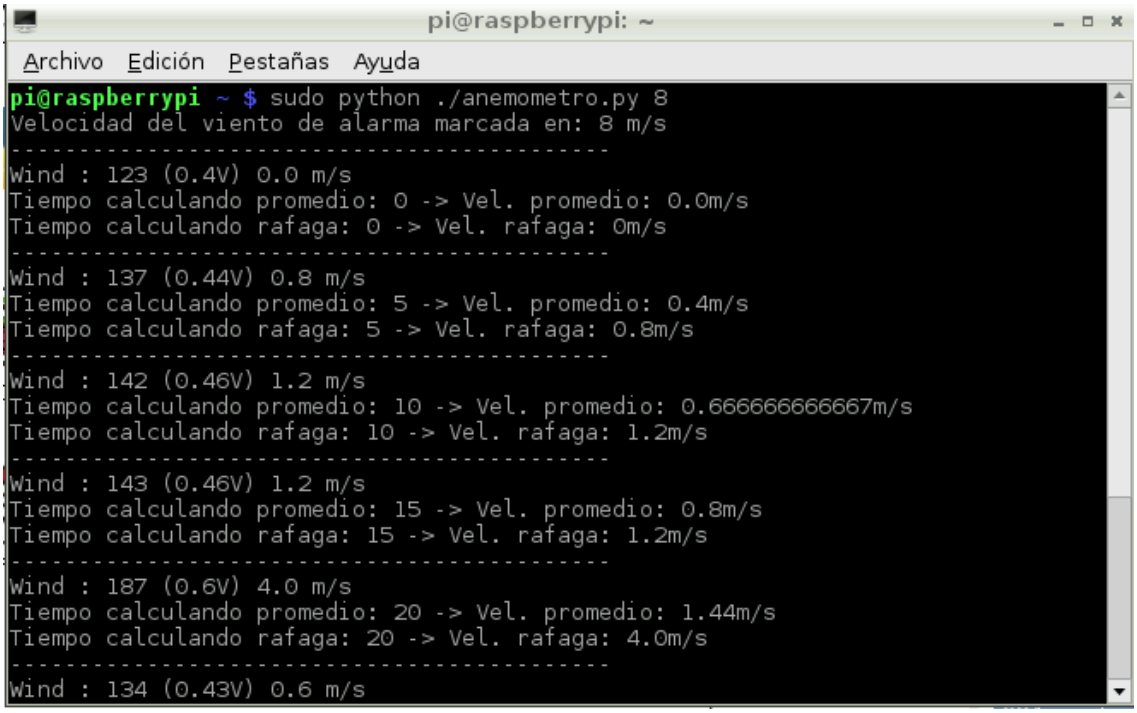


Ilustración 34 – Ejecución del programa anemómetro.py

6.3. Sistema de monitorización de la temperatura y humedad

Como explicamos en el apartado de metodología, vamos a utilizar el sensor AM2315 para medir la temperatura y humedad del ambiente. Este sensor usa como interfaz de entrada de datos el bus I2C de 7 bits, que usa como dirección la 05C.

Haciendo una breve introducción al interfaz I2C, este fue diseñado por Philips a principios de los 80 para permitir una comunicación sencilla entre componentes que se encontraban en la misma placa circuital. El nombre I2C proviene de “Inter IC”, y en algunas ocasiones se le llama IIC o bus I^2C. Originalmente se diseñó con una velocidad máxima de 100 kbit por segundo; actualmente existe un modo para dispositivos que requieren mayores velocidades, el “fast mode plus”, pero este ya deja de ser en cierta manera un bus I2C. Algunos de sus elementos más significativos son:

- Requiere 2 líneas de bus.
- No necesita estrictamente un ratio de baudios, ya que el dispositivo maestro genera un bus de reloj.

- Contiene un sistema de maestro/esclavo sencillo entre todos los componentes. Cada dispositivo conectado al bus se le proporciona una dirección única.

- El I2C es un auténtico bus "multi-master", ya que posee un sistema de detección de colisiones.

Una vez aclarado esto, detallamos el cableado que va a llevar el sensor. El AM2315 posee 4 cables, el cable rojo va a la alimentación de 5V de la RPi, el cable negro a GND, el cable amarillo al pin SDA y el cable blanco al SCL. En el siguiente dibujo podemos verlo gráficamente:

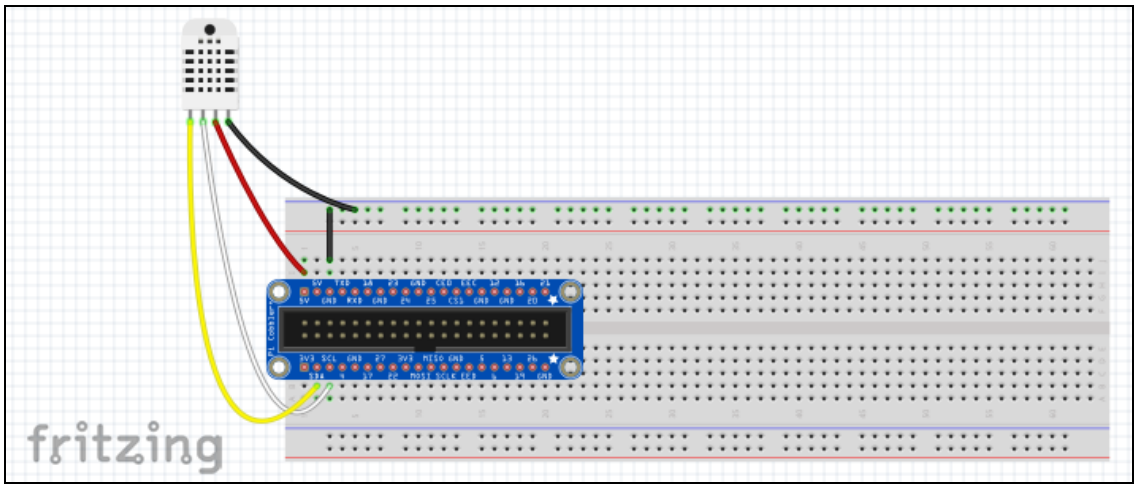


Ilustración 35 – Esquema gráfico de la conexión del sensor AM2315 con la RPi
(*) Dado que no había una miniatura del AM2315, he usado para representarlo el sensor DHT22.

La siguiente gráfica nos muestra el esquema eléctrico:

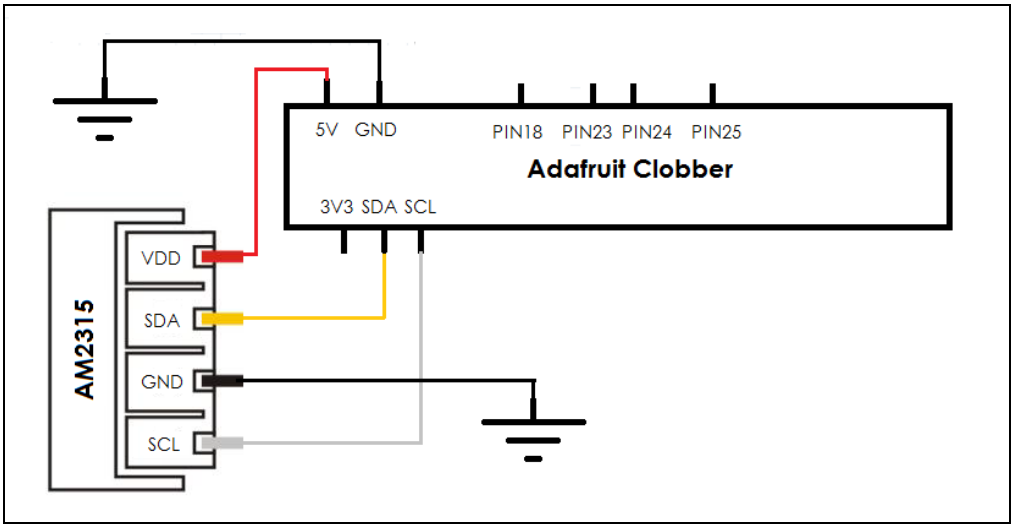


Ilustración 36 – Esquema eléctrico de la conexión del sensor AM2315 con la RPi

Con el sensor ya cableado, necesitamos configurar el bus I2C en la Raspberry Pi. Para ello emplearemos los siguientes comandos:

```
sudo apt-get install python-smbus
```

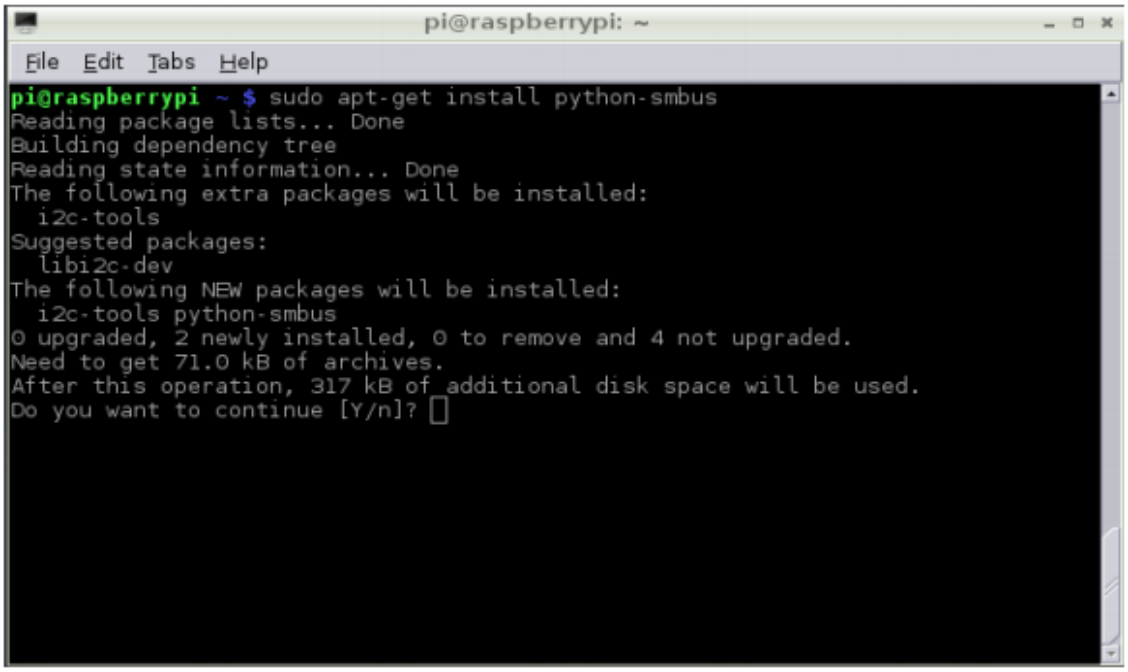


Ilustración 37 – Instalación del bus I2C en la RPi (I)

```
sudo add user pi i2c
```

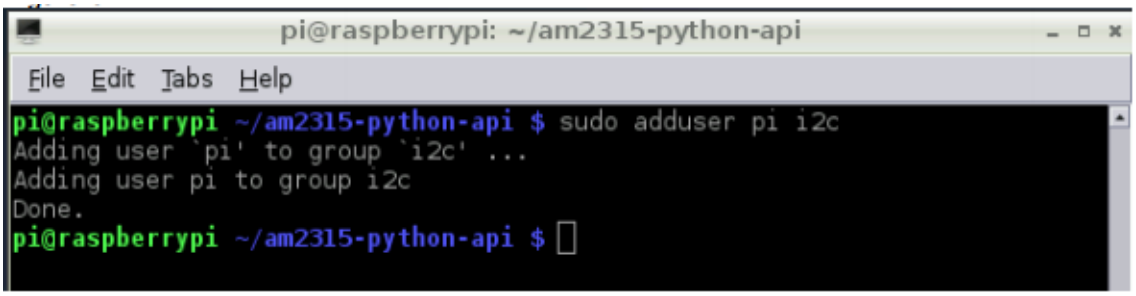


Ilustración 38 - Instalación del bus I2C en la RPi (II)

```
sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

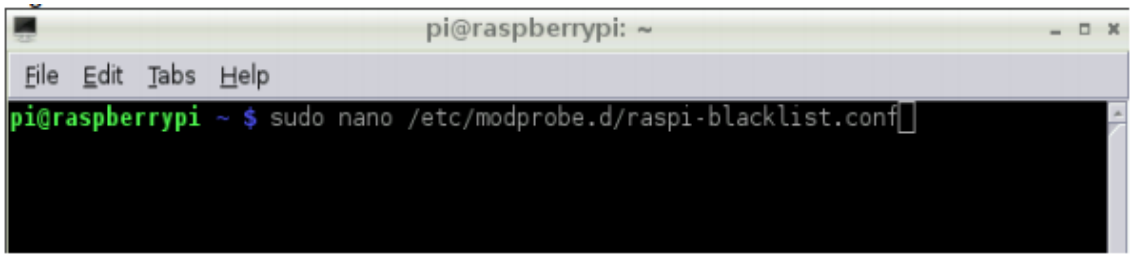


Ilustración 39 - Instalación del bus I2C en la RPi (III)

Ahora ponemos un # en la línea que contiene blacklist i2c-bcm2708. Luego guardamos y cerramos el fichero (Ctrl-O, Ctrl-X).

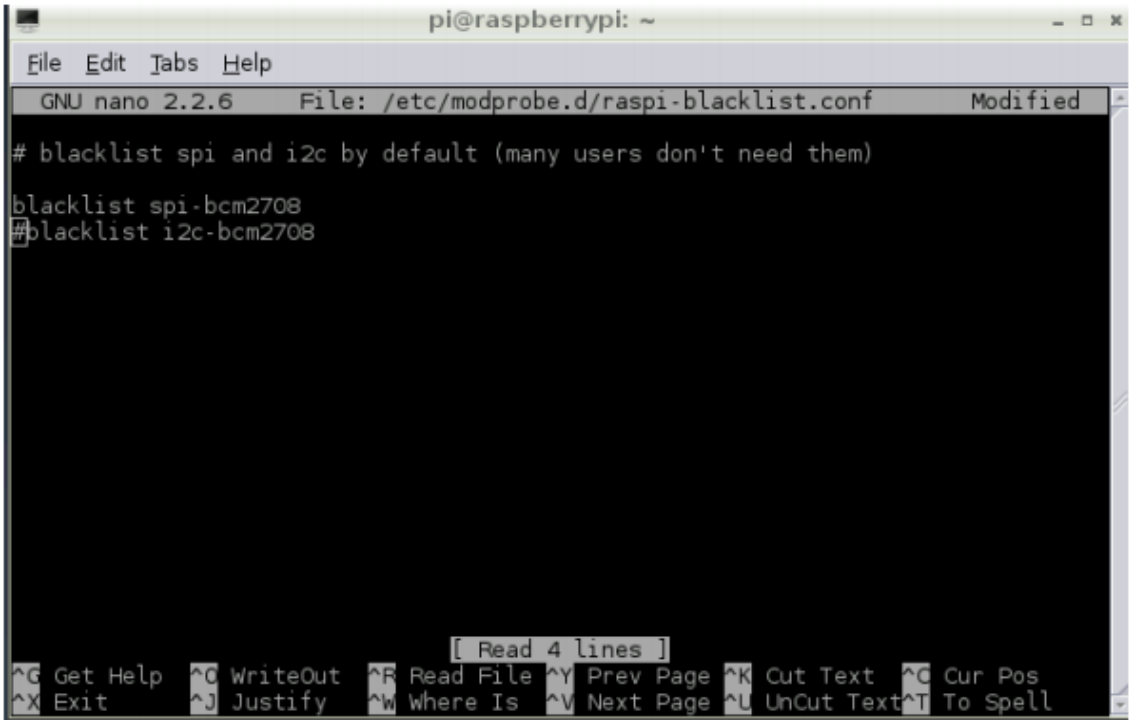


Ilustración 40 - Instalación del bus I2C en la RPi (IV)

```
sudo nano /etc/modules
```

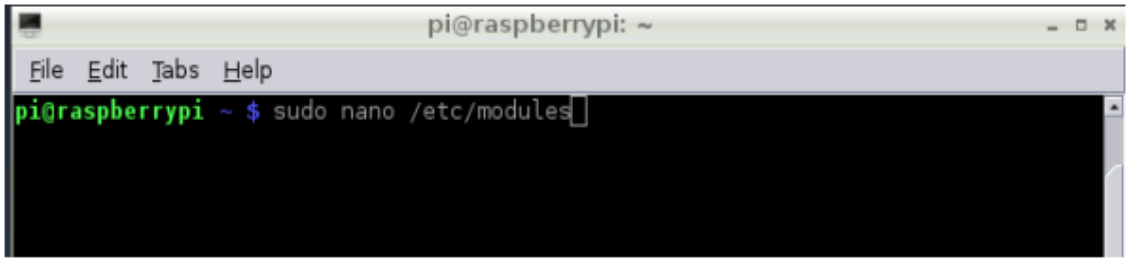


Ilustración 41 - Instalación del bus I2C en la RPi (V)

Añadimos las siguientes líneas al archivo abierto:

i2c-dev

i2c-bcm2708

Tras esto guardamos y cerramos el fichero (Ctrl-O, Ctrl-X).

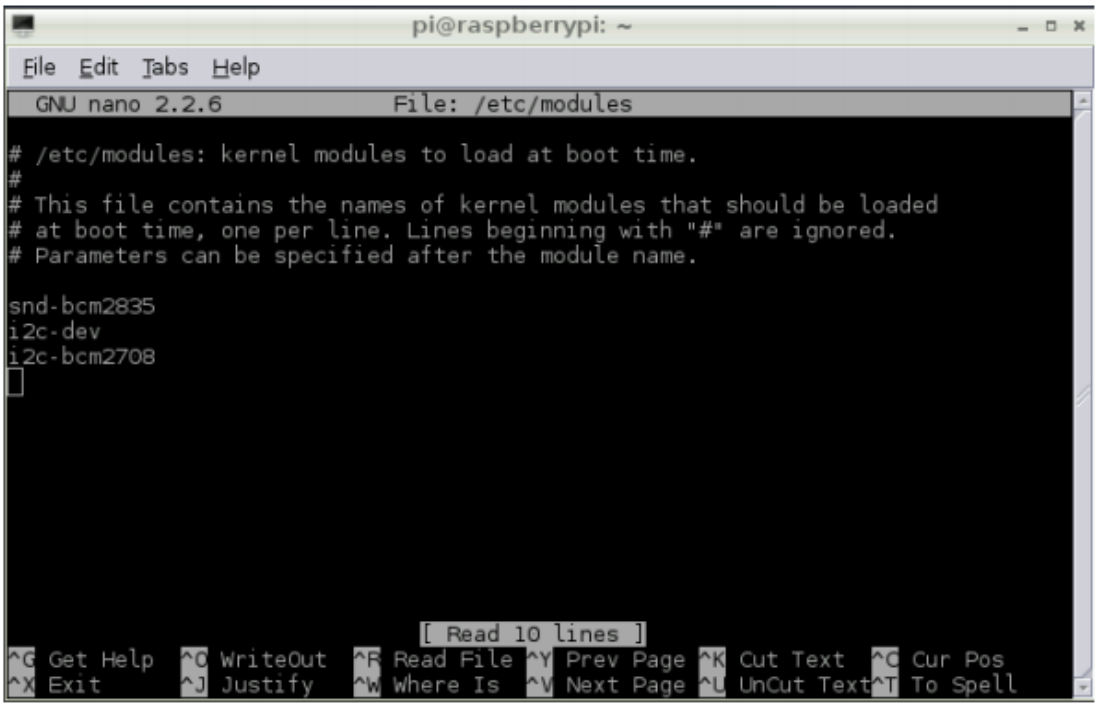


Ilustración 42- Instalación del bus I2C en la RPi (VI)

Ahora reiniciamos la RPi para que fije todos los cambios realizados:

```
sudo reboot
```

Una vez está configurado el bus I2C, vamos a comprobar que este detecta el sensor conectado. Para ello utilizaremos el comando:

```
sudo i2cdetect -y 1
```

Cabe destacar que tendremos que ejecutar dos veces este comando para probar que realmente detecta el dispositivo. Esto se debe a que el AM2315 se encuentra por defecto en "modo dormir", de manera que la primera vez que lo llamamos sirve para "despertarlo" y la segunda ya aparece activo.

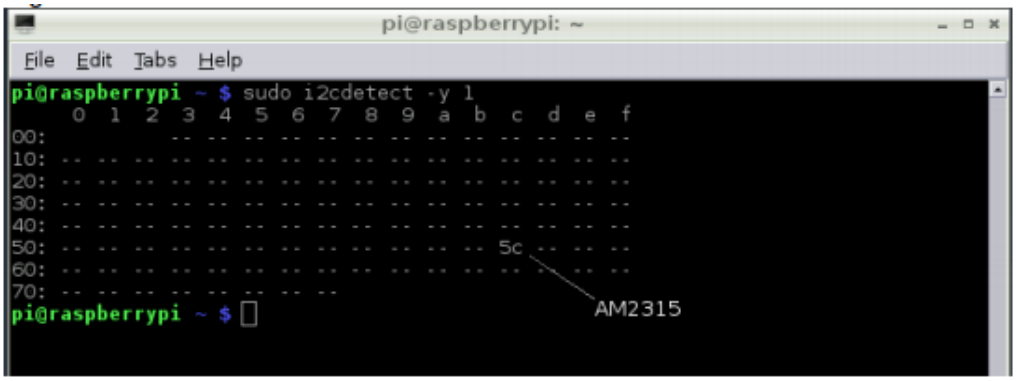


Ilustración 43 - Instalación del bus I2C en la RPi (VII)

Una vez configurado todo correctamente podemos pasar a escribir el código que leerá la temperatura y la humedad del ambiente.

```
#!/usr/bin/python

import time
import io
import fcntl

I2C_SLAVE=0x0703
I2C_BUS=1
AM2315_ADDR = 0x5c
CMD_READ = "\x03"

class i2c(object):
    def __init__(self, device, bus):
        self.fr = io.open("/dev/i2c-"+str(bus), "rb", buffering=0)
        self.fw = io.open("/dev/i2c-"+str(bus), "wb", buffering=0)
        fcntl.ioctl(self.fr, I2C_SLAVE, device)
        fcntl.ioctl(self.fw, I2C_SLAVE, device)

    def writeI2C(self, bytes):
        self.fw.write(bytes)

    def readI2C(self, bytes):
        return self.fr.read(bytes)

    def closeI2C(self):
        self.fw.close()
        self.fr.close()

am = i2c(AM2315_ADDR, I2C_BUS) # AM2315 0x5C, bus 1

try:
    am.writeI2C(CMD_READ) # Envia una señal de lectura para
“despertarlo”
except IOError:
    pass # Esto generara un ioError, ya que AM2315 no responde
mientras se despierta, por tanto se ingnora el error para leer
time.sleep(0.1)
am.writeI2C(CMD_READ+"\x00\x04") # envia un comando de lectura
comenzando en cero (0) (0x00) y toma cuatro (4) (0x04) registros.
# Devuelve 8 bytes en total.
time.sleep(0.01)
data = am.readI2C(8) # Lee 8 bytes como datos. Solo los lee.
am.closeI2C()

s = bytearray(data) # Sacamos los valores del archivo de lectura
hum = (256 * s[2] + s[3]) / 10
temp = (256 * s[4] + s[5]) / 10
print "Temperatura: " + str(temp) + " C"
print "Humedad: " + str(hum) + "%"
```

Además, vamos a incluirle el cálculo de la temperatura de rocío, el cual se realiza con la siguiente fórmula:

$$Pr = \sqrt[8]{\frac{H}{100}} \cdot (112 + 0.9 \cdot T) + (0.1 \cdot T) - 112 \quad (1)$$

Con el punto de rocío podremos establecer una alarma de temperatura, de manera que si la temperatura de ambiente es menor que la de rocío saltará un mensaje de advertencia por condensación del agua del aire.

```
raiz = hum/float(100)
P_rocio = pow(raiz, 1.0/8) * (112 + (0.9*temp)) + (0.1*temp) - 112

print "Punto de rocio: " + str(P_rocio) + " C"

if int(temp) <= int(P_rocio): #Si la temperatura es menor que la
del punto de rocio hay condensacion
        print "Punto de rocio alcanzado! No se puede obervar"
else:
        print "Condiciones optimas de Temp y Humedad"
```

Con esto ya podemos escribir el código completo:

```
#!/usr/bin/python

import time
import io
import fcntl

I2C_SLAVE=0x0703
I2C_BUS=1
AM2315_ADDR = 0x5c
CMD_READ = "\x03"

class i2c(object):
    def __init__(self, device, bus):
        self.fr = io.open("/dev/i2c-"+str(bus), "rb", buffering=0)
        self.fw = io.open("/dev/i2c-"+str(bus), "wb", buffering=0)
        fcntl.ioctl(self.fr, I2C_SLAVE, device)
        fcntl.ioctl(self.fw, I2C_SLAVE, device)

    def writeI2C(self, bytes):
        self.fw.write(bytes)

    def readI2C(self, bytes):
        return self.fr.read(bytes)
```

```
    def closeI2C(self):
        self.fw.close()
        self.fr.close()

am = i2c(AM2315_ADDR, I2C_BUS) # AM2315 0x5C, bus 1

try:
    am.writeI2C(CMD_READ) # Envia una señal de lectura para
“despertarlo”
except IOError:
    pass # Esto generara un ioError, ya que AM2315 no responde
mientras se despierta, por tanto se ingnora el error para leer

time.sleep(0.1)
am.writeI2C(CMD_READ+"\x00\x04") # envia un comando de lectura
comenzando en cero (0) (0x00) y toma cuatro (4) (0x04) registros.
# Devuelve 8 bytes en total.

time.sleep(0.01)
data = am.readI2C(8) # Lee 8 bytes como datos. Solo los lee.
am.closeI2C()

s = bytearray(data) # Sacamos los valores del archivo de lectura
hum = (256 * s[2] + s[3]) / 10
temp = (256 * s[4] + s[5]) / 10

raiz = hum/float(100)
P_rocio = pow(raiz, 1.0/8) * (112 + (0.9*temp)) + (0.1*temp) - 112

print "Temperatura: " + str(temp) + " C"
print "Humedad: " + str(hum) + "%"
print "Punto de rocio: " + str(P_rocio) + " C"

if int(temp) <= int(P_rocio): #Si la temperatura es menor que la
del punto de rocio hay condensacion
        print "Punto de rocio alcanzado! No se puede obervar"
else:
        print "Condiciones optimas de Temp y Humedad"
```

Finalmente, haremos una demostración del funcionamiento de nuestro código:

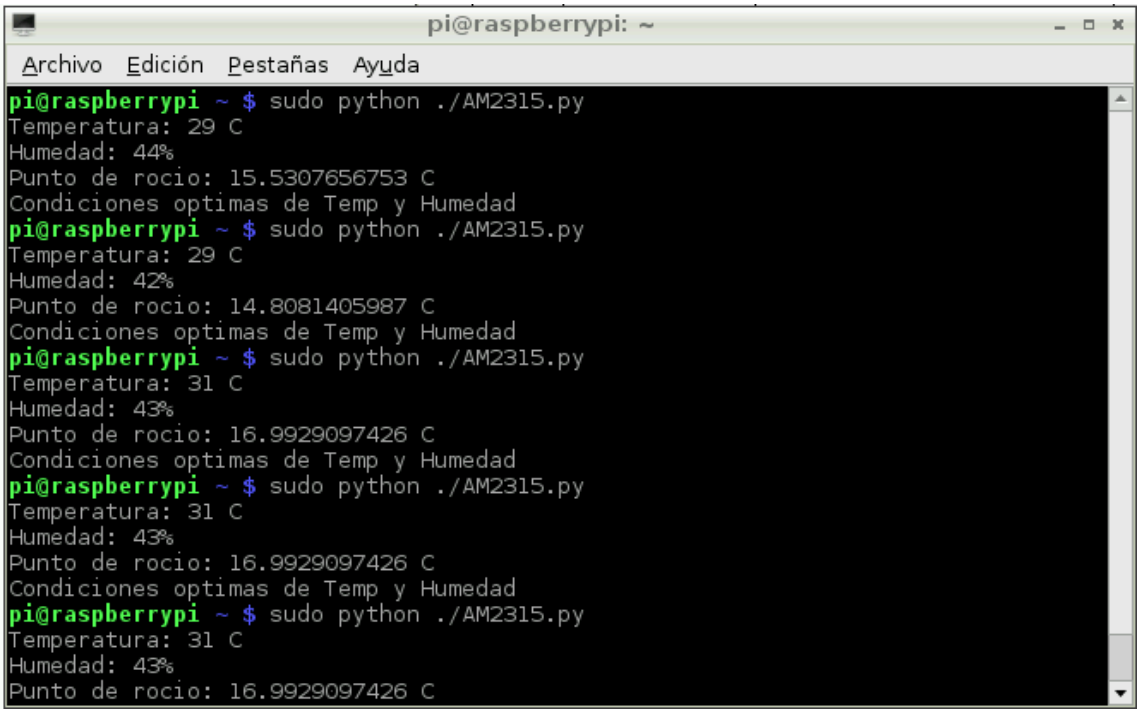


Ilustración 44 – Ejecución del programa AM2315.py

6.4. Unificación de los códigos de los sensores

Con los códigos del anemómetro y del AM2315 funcionando correctamente, debemos unificarlos para obtener un único programa ejecutándose. Para ello, deben copiarse ordenadamente en un fichero en blanco todos los elementos:

1) Los paquetes a importar

2) La definición de variables y clases.

3) El bucle, el código que se ejecuta cada 5 segundos.

Uno de los problemas que encontramos es que tenemos para cada programa un tiempo de retardo diferente; lo que vamos a hacer es unificar ambos a 5 segundos, ya que estos dos segundos dependen de los contadores creados para calcular la velocidad de viento promedio y la velocidad de ráfaga.

Dicho esto, procedemos a representar el código unificado correctamente:

```
#!/usr/bin/python

import time
import io
import fcntl

import os
import RPi.GPIO as GPIO
import sys

# Variables del AM2315
I2C_SLAVE=0x0703
I2C_BUS=1
AM2315_ADDR = 0x5c
CMD_READ = "\x03"


# Variables del anemometro
GPIO.setmode(GPIO.BCM)
# Pines conectados desde el puerto SPI en el ADC hasta el Cobbler
SPICLK = 18
SPIMISO = 23
SPIMOSI = 24
SPICS = 25

# Configuramos los pines  de la interfaz SPI
GPIO.setup(SPIMOSI, GPIO.OUT)
GPIO.setup(SPIMISO, GPIO.IN)
GPIO.setup(SPICLK, GPIO.OUT)
GPIO.setup(SPICS, GPIO.OUT)

# La salida del anemometro OUTPUT conectada al ADC #0
wind_channel = 0;
DEBUG = 1
WindWarning = 8 # Valor por defecto de la velocidad del viento
alarma

count1 = 0 # inicializo contador 1 (vel. promedio)
count1_trigg = 600 # 10 min * 60 seg = 600 seg
count2 = 0 # inicializo contador 2 (vel. rafaga)
count2_trigg = 3600 # 1h * 60 min * 60seg = 3600 seg
windsum = 0 # inicializo la variable sumatorio
windraf = 0 # inicializo la variable de ráfaga

#Clases del AM2315
class i2c(object):
    def __init__(self, device, bus):
        self.fr = io.open("/dev/i2c-"+str(bus), "rb", buffering=0)
        self.fw = io.open("/dev/i2c-"+str(bus), "wb", buffering=0)
        fcntl.ioctl(self.fr, I2C_SLAVE, device)
        fcntl.ioctl(self.fw, I2C_SLAVE, device)
```

```
    def writeI2C(self, bytes):
        self.fw.write(bytes)

    def readI2C(self, bytes):
        return self.fr.read(bytes)

    def closeI2C(self):
        self.fw.close()
        self.fr.close()

am = i2c(AM2315_ADDR, I2C_BUS) # AM2315 0x5C, bus 1

#Clases del anemometro
# Lee los datos SPI del chip MCP3008, 8 posibles adc's (0 hasta 7)
def readadc(adcnum, clockpin, mosipin, misopin, cspin):
        if ((adcnum > 7) or (adcnum < 0)):
                return -1
        GPIO.output(cspin, True)

        GPIO.output(clockpin, False)  # empieza el clck en baja
        GPIO.output(cspin, False)     # empieza el  CS en baja

        commandout = adcnum
        commandout |= 0x18  # bit de inicio + bit unico de fin
        commandout <<= 3    # solo necesitamos enviar 5 bits

        for i in range(5):
                if (commandout & 0x80):
                        GPIO.output(mosipin, True)
                else:
                        GPIO.output(mosipin, False)
                commandout <<= 1
                GPIO.output(clockpin, True)
                GPIO.output(clockpin, False)

        adcout = 0

    # lee un bit vacio, un bit null y 10 ADC bits
        for i in range(12):
                GPIO.output(clockpin, True)
                GPIO.output(clockpin, False)
                adcout <<= 1
                if (GPIO.input(misopin)):
                        adcout |= 0x1

        GPIO.output(cspin, True)

        adcout >>= 1       # el primer bit es 'null', lo eliminamos
        return adcout
```

```
# Funcion para convertir los datos en nivel de voltaje,
# redondeado al numero de decimales que espedificaquemos (places).
def ConvertVolts(adcout,places):
        volts = (adcout * 3.3) / float(1023)
        volts = round(volts,places)
        return volts

# Funcion para calcular el viento de los datos del Anemometro
# redondeado al numero de decimales que espedificaquemos (places).
def ConvertWind(volts,places):

# Rango de voltaje del anemometro (0.4V-2.0V)
# cualquier valor debajo de 0.4V se considerara como 0 m/s
# Si evaluas 0.4 como 0 y 2.0 como 32 el ratio es linear a 2 m/s
por cada 100mV
  if volts <= 0.4:
        wind = 0

  else:
        wind = (volts - 0.4) * 10 * 2

  wind = round(wind,places)
  return wind

def warnwind():
    warn = input("Introduzca el nuevo valor del Alarma por viento:
")
    WindWarning = warn
    print "Velocidad del viento de alarma marcada en: " +
str(WindWarning) + " m/s"


# Ejecucion del programa
while True:
        print "-------------------------------------------"
        # AM2315 part
        try:
            am.writeI2C(CMD_READ) # Envia una señal de lectura para
“despertarlo”
        except IOError:
            pass # Esto generara un ioError, ya que AM2315 no
responde mientras se despierta, por tanto se ingnora el error para
leer

        time.sleep(0.1)
        am.writeI2C(CMD_READ+"\x00\x04") # envia un comando de
lectura comenzando en cero (0) (0x00) y toma cuatro (4) (0x04)
registros.
        # Devuelve 8 bytes en total.

        time.sleep(0.01)
        data = am.readI2C(8) # Lee 8 bytes como datos. Solo los lee.
        s = bytearray(data) # Sacamos los valores del archivo
```

```
        hum = (256 * s[2] + s[3]) / 10
        temp = (256 * s[4] + s[5]) / 10

        raiz = hum/float(100)
        P_rocio = pow(raiz, 1.0/8) * (112 + (0.9*temp)) +
(0.1*temp) - 112

        print "Temperatura: " + str(temp) + " C"
        print "Humedad: " + str(hum) + "%"
        print "Punto de rocio: " + str(P_rocio) + " C"

        if int(temp) <= int(P_rocio): #Si la temperatura es menor
que la del punto de rocio hay condensacion
            print "Punto de rocio alcanzado! No se puede obervar"
        else:
            print ""

        # ANEMOMETER PART
        # Lee el pin analogico
        wind_level = readadc(wind_channel, SPICLK, SPIMOSI,
SPIMISO, SPICS)
        wind_volts = ConvertVolts(wind_level,2)
        wind = ConvertWind(wind_volts,2)
        windsum = windsum + wind

        if float(wind) > float(windraf):
                windraf = wind

        windprom = windsum / ((count1 + 5) / 5)

        if DEBUG:
            print("Wind : {} ({}V) {}
      m/s".format(wind_level,wind_volts,wind))
            print("Tiempo calculando promedio: " + str(count1) + "
-> Vel. promedio: " + str(windprom) + "m/s")
            print("Tiempo calculando rafaga: " + str(count2) + " ->
Vel. rafaga: " + str(windraf) + "m/s")
            if wind >= WindWarning:
                    print "Alerta! Cerrar cupula por viento"

        count1 = count1 + 5
        count2 = count2 + 5

        if int(count1) == int(count1_trigg):
                count1 = 0 # vuelvo a inicializar el contador
                windsum = 0 # inicializo sumatorio

        if int(count2) == int(count2_trigg):
                count2 = 0 # vuelvo a inicializar el contador
                windraf = 0 # inicializo el valor de rafaga

        # Esperamos 5 segundos a que reinicie el bucle
        time.sleep(5)
```

Finalmente, comprobamos el correcto funcionamiento del programa:

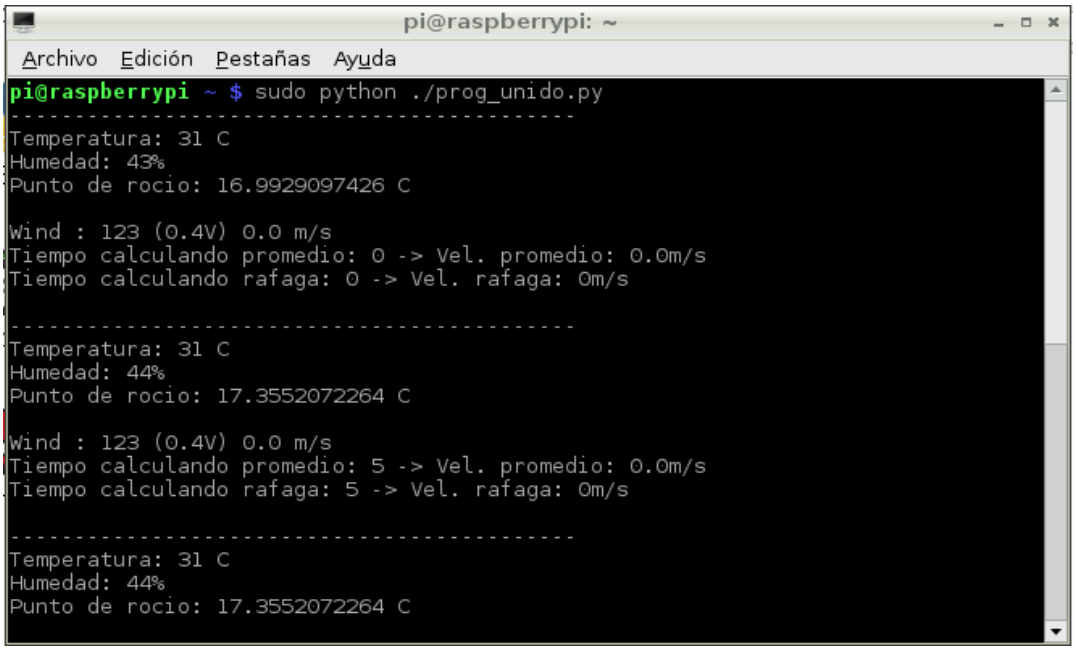


Ilustración 45 – Ejecución del programa prog_unido.py

6.5. Servidor TCP-IP y comunicación con el ordenador

Para la comunicación de la Raspberry y Pi con el ordenador del telescopio se precisará un protocolo TCP-IP. Se escogió este ya que es sencillo y podremos acceder a los datos de la RPi desde cualquier dispositivo si ambos están conectados a la misma red.

El modelo TCP-IP describe un conjunto de guías generales de diseño e implementación de protocolos de red específicos para permitir que un equipo pueda comunicarse en una red. TCP/IP provee conectividad de extremo a extremo especificando como los datos deberían ser formateados, direccionados, transmitidos, enrutados y recibidos por el destinatario. El modelo TCP/IP y los protocolos relacionados son mantenidos por la Internet Engineering Task Force (IETF).

Para conseguir un intercambio fiable de datos entre dos equipos, se deben llevar a cabo muchos procedimientos separados. El resultado es que el software de comunicaciones es complejo. Con un modelo en capas o niveles resulta más sencillo agrupar funciones relacionadas e implementar el software modular de comunicaciones.

El proceso que aplicaremos en nuestro programa completo con los sensores de

temperatura, humedad y viento será añadirle un servidor TCP-IP, al cual puedan conectarse el resto de ordenadores como clientes para obtener los datos generados. El ordenador cliente mandará diferentes comandos para el envío de datos o para cambiar valores de las variables de velocidad del viento de alarma y los contadores para calcular la velocidad de ráfaga y promedio del viento.

El método de TCP-IP que utiliza Python es el estándar por medio de sockets, que para mantener la conexión con el cliente sin que se cierre cada vez que enviamos datos tendrá los llamados hilos o "threads". El siguiente código creará sockets para mantener a los diferentes clientes en línea con el servidor.

```
import socket
import struct
from thread import *

HOST = ''   # Valor simbolico para permitir a cualquier cliente
PORT = 1031 # Puerto TCP con el que nos conectaremos

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

print 'Socket creado'

# Bindeamos el socket al Puerto y el host local
try:
    s.bind((HOST, PORT))
except socket.error , msg:
    print 'Bind failed. Error Code : ' + str(msg[0]) + ' Message '
+ msg[1]
    sys.exit()

print 'Socket enlazado correctamente'

# Comienza a escuchar el socket a los clientes
s.listen(5)
print 'Socket escuchando...'

# Funcion para mantener conexiones. Se usara para crear threads
def clientthread(conn):
    # Enviamos un mensaje al cliente conectado
    conn.send('Bienvenido al servidor. Escribe el comando deseado y
pulsa enter:\n') # send solo envía strings
    reply = ""
    # Bucle infinito que permite al thread mantenerse en el tiempo.
    while True:
        # Recibo comando del cliente
        data = conn.recv(1024)
        if not data:
            break
```

```
        conn.sendall(reply)
        print (reply)
    # Salimos del bucle
    conn.close()

# Ahora seguimos conectados con el cliente
while 1:

        # Esperamos a aceptar la conexion - llamada de bloqueo
        conn, addr = s.accept()
        print 'Connected with ' + addr[0] + ':' + str(addr[1])

        # Comenzamos un Nuevo thread. El 1er argumento es la
funcion que se llama, el segundo es la tupla de argumentos de la
funcion.
        start_new_thread(clientthread ,(conn,))

s.close()
```

Para comprobar el correcto funcionamiento del programa podemos realizar un Telnet desde el ordenador cliente para conectar al servidor de la Raspberry Pi. En mi caso me he conectado mediante telnet con la misma RPi.

Telnet es una herramienta de línea de comandos que está diseñada para la administración de servidores remotos a través del Símbolo del sistema (Command Prompt).

Para instalarlo correctamente abrimos una nueva ventana de LXTerminal y ponemos el siguiente comando:

```
sudo apt-get install telnet
```

Para usar Telnet simplemente debemos escribir: `telnet [dirección] [puerto]`

Una vez configurado el cliente Telnet podemos mostrar cómo queda el programa completo de Python.

```
#!/usr/bin/python

import time
import io
import fcntl

import os
import RPi.GPIO as GPIO
```

```
import sys

import socket
import struct
from thread import *

# Variables del AM2315
I2C_SLAVE=0x0703
I2C_BUS=1
AM2315_ADDR = 0x5c
CMD_READ = "\x03"

# Variables del anemometro
GPIO.setmode(GPIO.BCM)
# Pines conectados desde el puerto SPI en el ADC hasta el Cobbler
SPICLK = 18
SPIMISO = 23
SPIMOSI = 24
SPICS = 25

# Configuramos los pines  de la interfaz SPI
GPIO.setup(SPIMOSI, GPIO.OUT)
GPIO.setup(SPIMISO, GPIO.IN)
GPIO.setup(SPICLK, GPIO.OUT)
GPIO.setup(SPICS, GPIO.OUT)

# La salida del anemometro OUTPUT conectada al ADC #0
wind_channel = 0;
DEBUG = 1
WindWarning = 8 # Valor por defecto de la velocidad del viento
alarma

count1 = 0 # inicializo contador 1 (vel. promedio)
count1_trigg = 600 # 10 min * 60 seg = 600 seg
count2 = 0 # inicializo contador 2 (vel. rafaga)
count2_trigg = 3600 # 1h * 60 min * 60seg = 3600 seg
windsum = 0 # inicializo la variable sumatorio
windraf = 0 # inicializo la variable de ráfaga

#Clases del AM2315
class i2c(object):
    def __init__(self, device, bus):
        self.fr = io.open("/dev/i2c-"+str(bus), "rb", buffering=0)
        self.fw = io.open("/dev/i2c-"+str(bus), "wb", buffering=0)
        fcntl.ioctl(self.fr, I2C_SLAVE, device)
        fcntl.ioctl(self.fw, I2C_SLAVE, device)

    def writeI2C(self, bytes):
        self.fw.write(bytes)

    def readI2C(self, bytes):
        return self.fr.read(bytes)
```

```
    def closeI2C(self):
        self.fw.close()
        self.fr.close()

am = i2c(AM2315_ADDR, I2C_BUS) # AM2315 0x5C, bus 1

#Clases del anemometro
# Lee los datos SPI del chip MCP3008, 8 posibles adc's (0 hasta 7)
def readadc(adcnum, clockpin, mosipin, misopin, cspin):
        if ((adcnum > 7) or (adcnum < 0)):
                return -1
        GPIO.output(cspin, True)

        GPIO.output(clockpin, False)  # empieza el clck en baja
        GPIO.output(cspin, False)     # empieza el  CS en baja

        commandout = adcnum
        commandout |= 0x18  # bit de inicio + bit unico de fin
        commandout <<= 3    # solo necesitamos enviar 5 bits

        for i in range(5):
                if (commandout & 0x80):
                        GPIO.output(mosipin, True)
                else:
                        GPIO.output(mosipin, False)
                commandout <<= 1
                GPIO.output(clockpin, True)
                GPIO.output(clockpin, False)

        adcout = 0

    # lee un bit vacio, un bit null y 10 ADC bits
        for i in range(12):
                GPIO.output(clockpin, True)
                GPIO.output(clockpin, False)
                adcout <<= 1
                if (GPIO.input(misopin)):
                        adcout |= 0x1

        GPIO.output(cspin, True)

        adcout >>= 1       # el primer bit es 'null', lo
eliminamos
        return adcout

# Funcion para convertir los datos en nivel de voltaje,
# redondeado al numero de decimales que especifiquemos (places).
def ConvertVolts(adcout,places):
        volts = (adcout * 3.3) / float(1023)
        volts = round(volts,places)
        return volts
```

```
# Funcion para calcular el viento de los datos del Anemometro
# redondeado al numero de decimales que especifiquemos (places).
def ConvertWind(volts,places):

# Rango de voltaje del anemometro (0.4V-2.0V)
# cualquier valor debajo de 0.4V se considerara como 0 m/s
# Si evaluas 0.4 como 0 y 2.0 como 32 el ratio es linear a 2 m/s
por cada 100mV
  if volts <= 0.4:
        wind = 0

  else:
        wind = (volts - 0.4) * 10 * 2

  wind = round(wind,places)
  return wind

# Elementos para la comunicacion TCP/IP

HOST = ''   # Valor simbolico para permitir a cualquier cliente
PORT = 1031 # Puerto TCP con el que nos conectaremos

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

print 'Socket creado'

# Bindeamos el socket al Puerto y el host local
try:
    s.bind((HOST, PORT))
except socket.error , msg:
    print 'Bind failed. Error Code : ' + str(msg[0]) + ' Message '
+ msg[1]
    sys.exit()

print 'Socket enlazado correctamente'

# Comienza a escuchar el socket a los clientes
s.listen(5)
print 'Socket escuchando...'

# Funcion para mantener conexiones. Se usara para crear threads
def clientthread(conn):
    # Enviamos un mensaje al cliente conectado
    conn.send('Bienvenido al servidor. Escribe el comando deseado y
pulsa enter:\n') # send solo envía strings
    reply = ""

    # Bucle infinito que permite al thread mantenerse en el tiempo.
    while True:
        # Recibo comando del cliente
        data = conn.recv(1024)
        word = data.split()
```

```
        if word[0] == "send": #Si envio "send" me manda todos los
datos y posibles alarmas
            reply = ("Temp: " + str(temp) + "\n"
                     "Hum: " + str(hum) + "\n"
                     "Rocio: " + str(P_rocio) + "\n"
                     "Vviento: " + str(wind) + "\n"
                     "Vprom: " + str(windprom) + "\n"
                     "Vraf: " + str(windraf) + "\n"
                     "AlarmaTemp/Hum: " + A_humtemp + "\n"
                     "AlarmaViento: " + A_viento + "\n")

        if word[0] == "x": #Si envio "x", cierra la conexion
            break

        if word[0] == "setmaxwind": #Si envio "setmaxwind" cambio
la velocidad de alarma del viento
            data2 = word[1]
            stringdata2 = data2.decode('utf-8')
            int_data2 = int(stringdata2)
            WindWarning = int_data2
            reply = ("NuevaAlarmaViento: " + str(WindWarning) +
"\n")

        if word[0] == "setraf": #Si envio "setraf" cambio el tiempo
del contador de rafaga (por defecto es 3600)
            data2 = word[1]
            stringdata2 = data2.decode('utf-8')
            int_data2 = int(stringdata2)
            count2_trigg = int_data2
            reply = (" NuevoTiempoRafaga: " + str(count2_trigg) +
"\n")

        if word[0] == "setprom": #Si envio "setprom" cambio el
tiempo del contador de velocidad promedio (por defecto es 600)
            data2 = word[1]
            stringdata2 = data2.decode('utf-8')
            int_data2 = int(stringdata2)
            count1_trigg = int_data2
            reply = ("NuevoTiempoPromedio: " + str(count1_trigg) +
"\n")

        if not data:
            break

        conn.sendall(reply)

    # Salimos del bucle
    conn.close()
```

```
# Ahora seguimos conectados con el cliente
while 1:
        # Esperamos a aceptar la conexion - llamada de bloqueo
        conn, addr = s.accept()
        print 'Connected with ' + addr[0] + ':' + str(addr[1])

        # Comenzamos un Nuevo thread. El 1er argumento es la
funcion que se llama, el segundo es la tupla de argumentos de la
funcion.
        start_new_thread(clientthread ,(conn,))

        while True:
            # AM2315 part
            try:
                  am.writeI2C(CMD_READ) # Envia una señal de lectura
para "despertarlo"
            except IOError:
                  pass # Esto generara un ioError, ya que AM2315 no
responde mientras se despierta, por tanto se ingnora el error para
leer

            time.sleep(0.1)
            am.writeI2C(CMD_READ+"\x00\x04") # envia un comando de
lectura comenzando en cero (0) (0x00) y toma cuatro (4) (0x04)
registros.
            # Devuelve 8 bytes en total.
            time.sleep(0.01)
            data = am.readI2C(8) # Lee 8 bytes como datos. Solo los
lee.

            s = bytearray(data) # Sacamos los valores del archivo
            hum = (256 * s[2] + s[3]) / 10
            temp = (256 * s[4] + s[5]) / 10

            raiz = hum/float(100)
            P_rocio = pow(raiz, 1.0/8) * (112 + (0.9*temp)) +
(0.1*temp) - 112

            if int(temp) <= int(P_rocio): #Si la temperatura es
menor que la del punto de rocio hay condensacion
                  A_humtemp = ("ON")
            else:
                  A_humtemp = ("OFF")

            # ANEMOMETER PART
            # Lee el pin analogico
            wind_level = readadc(wind_channel, SPICLK, SPIMOSI,
SPIMISO, SPICS)
            wind_volts = ConvertVolts(wind_level,2)
            wind = ConvertWind(wind_volts,2)
            windsum = windsum + wind
```

```
            if float(wind) > float(windraf):
                windraf = wind

            windprom = windsum / ((count1 + 5) / 5)

            if float(wind) >= float(WindWarning):
                A_viento = ("ON")
            else:
                A_viento = ("OFF")

        count1 = count1 + 5
        count2 = count2 + 5

        if int(count1) == int(count1_trigg):
                count1 = 0 # vuelvo a inicializar el contador
                windsum = 0 # inicializo sumatorio

        if int(count2) == int(count2_trigg):
                count2 = 0 # vuelvo a inicializar el contador
                windraf = 0 # inicializo el valor de rafaga

        # Esperamos 5 segundos a que reinicie el bucle
        time.sleep(5)

s.close()
```

Echando un vistazo más a fondo al código, vemos que entramos en un bucle tras aceptar la llamada de bloqueo del socket.accept() en el que se generarán todos los datos de los sensores cada 5 segundos como en nuestro programa anterior “prog_unido.py”.

Desde el ordenador del telescopio robótico, mediante un cliente de TCP, llamaremos a los comandos programados para que se le envíe la correspondiente información. También podremos cambiar algunos de los parámetros del programa desde el cliente como la velocidad máxima del viento a la que se producirá la alarma o el tiempo que precisa de cálculo la velocidad promedio y de ráfaga del viento. Los comandos que acepta la estación son los siguientes:

- “send”: Se solicita el envío los valores medidos más recientes por los sensores (se renuevan cada 5 segundos) y el estado de las alarmas de viento y temperatura. Es el comando más importante.

- “setmaxwind”: Asigna un nuevo valor a la velocidad del viento consigna con la que se activará la alarma de viento. Este valor está inicializado a 8 m/s.

- "setraf": Cambia el valor del contador que marca el tiempo de cálculo de la velocidad de ráfaga del viento. Esta variable está inicializada a 3600 (1 hora).

- "setprom": Cambia el valor del contador que marca el tiempo de cálculo de la velocidad promedio del viento. Esta variable está inicializada a 600 (10 minutos).

- "x": Este comando cierra la conexión entre el cliente y el servidor

Dado que la comunicación no se hará por medio de consola, sino que se ejecutará el programa cliente que pedirá los datos, los mensajes de respuesta deben ser cortos y claros, con una palabra que lo identifique y el valor en cuestión.

En caso de haber una alarma conectada (ON), el programa cliente del telescopio robótico mandará una instrucción al motor de la cúpula para que esta se cierre si estuviese abierta. Debido a la posibilidad de modificar el código fuente, el conjunto de comandos y funcionalidades puede ser fácilmente ampliado.

Una vez detallado el funcionamiento del código, comprobamos el correcto funcionamiento de todos los comandos mediante el cliente Telnet.

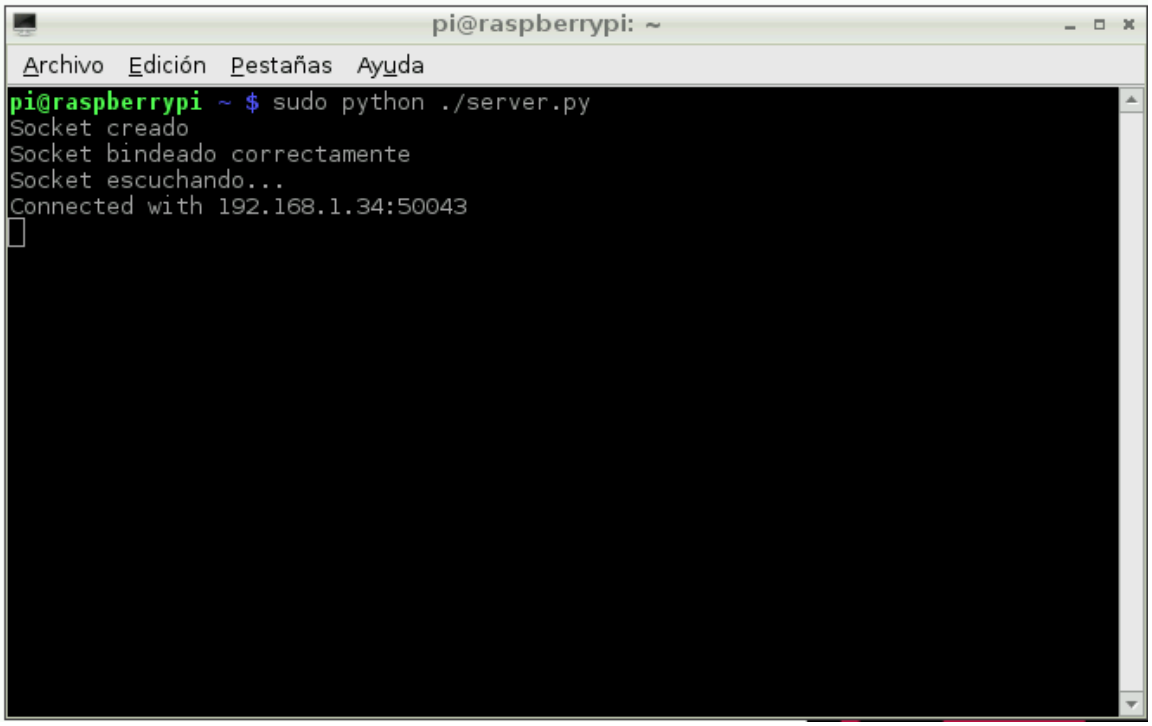


Ilustración 46 – Ejecución del programa server.py

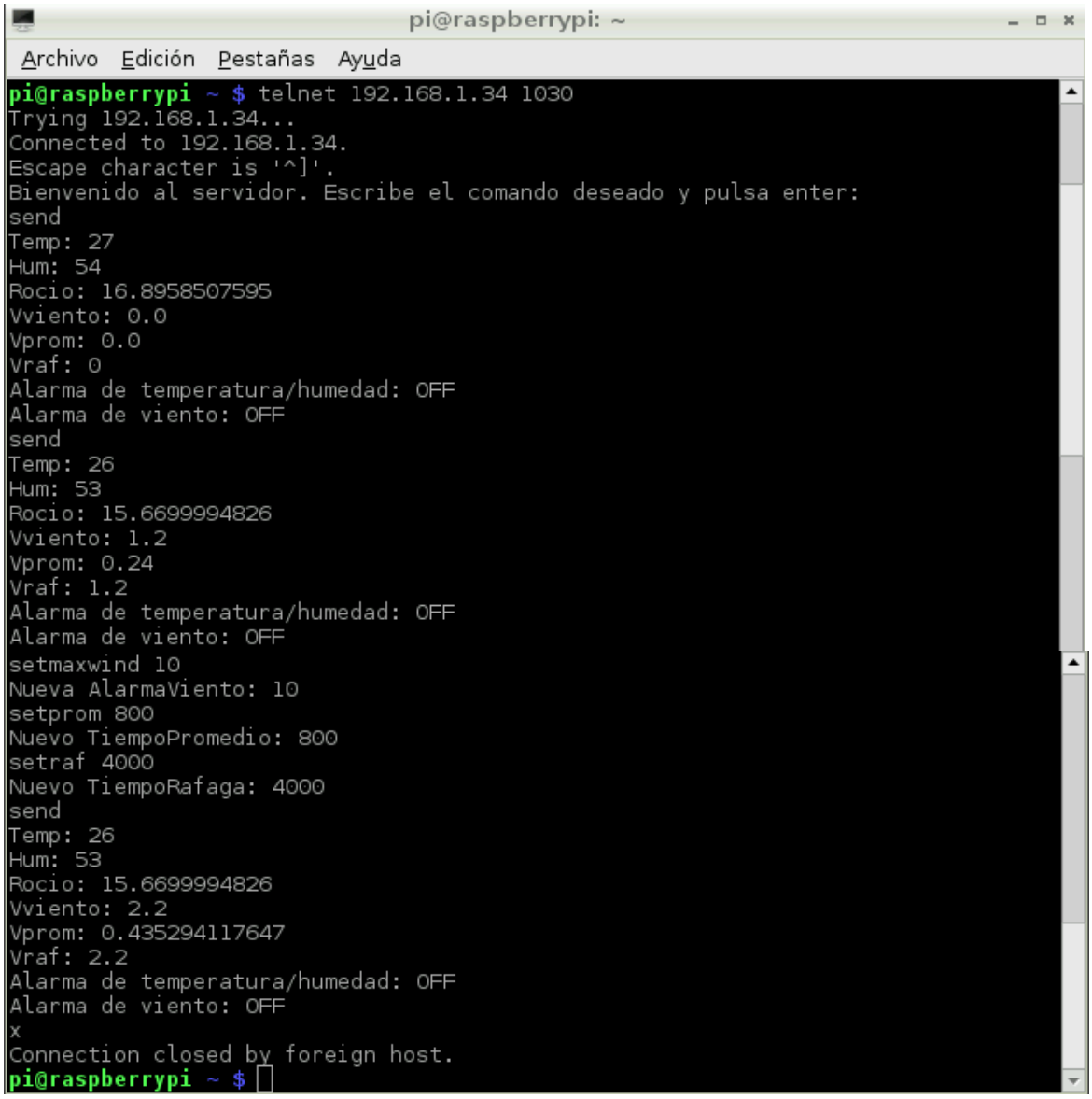


Ilustración 47 – Respuesta del cliente Telnet a server.py

6.6. Automatización del programa.

Como elemento final de nuestro proyecto, haremos que el programa diseñado en el apartado anterior se ejecute automáticamente cuando encendamos la Raspberry Pi.

Dado que la RPi va a estar conectada en red como elemento puente entre los sensores de temperatura, humedad y el anemómetro con el ordenador con Linux del telescopio robótico, no va a precisar de interfaz gráfica con la que interactuar, como una pantalla y un teclado. Lo único que necesitamos es que cada vez que esta se encienda ponga el funcionamiento nuestro programa, que se comunicará automáticamente con el ordenador mediante el protocolo TCP/IP. Una de las maneras más fáciles de hacer esto

es utilizando crontab.

Crontab es una tabla utilizada por cron, que es un daemon que se utiliza para ejecutar comandos específicos en un momento determinado. Crontab es muy flexible: se puede utilizar crontab para ejecutar un programa en el arranque o para repetir una tarea o programa a las 12 pm todos los miércoles, por ejemplo.

Para utilizar crontab con el Raspberry Pi para automatizar nuestro programa primeramente debemos hacer nuestro script de Python ejecutable. Haciendo click derecho en el icono y seleccionado la opción "propiedades"; en la pestaña de "Permisos", seleccionaremos la opción de "Hacer el fichero ejecutable" y guardamos los cambios en "Aceptar".

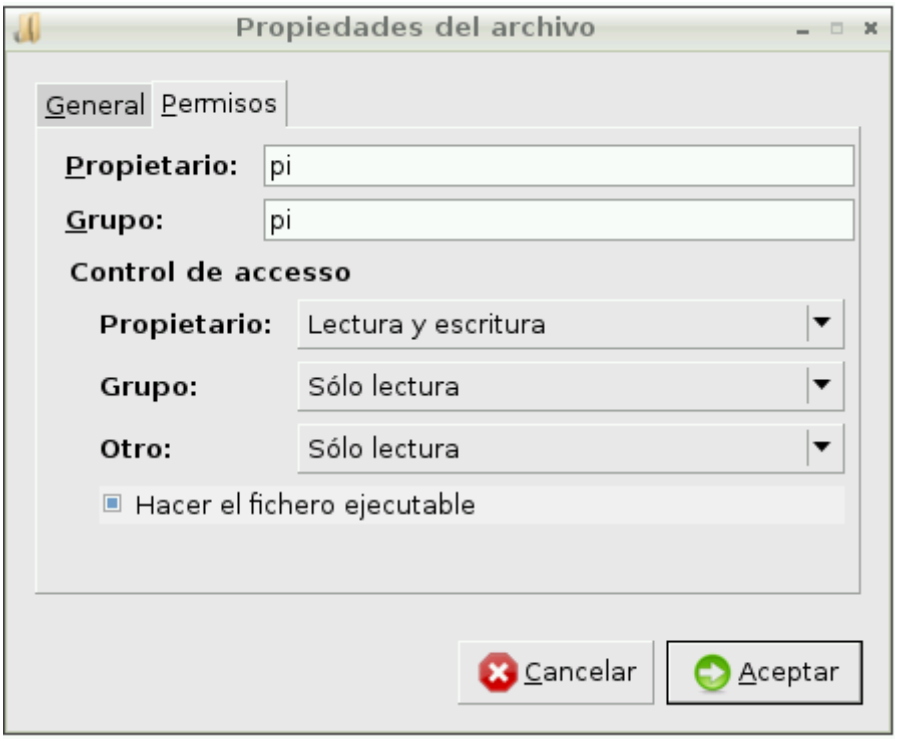


Ilustración 48 – Configuración de fichero ejecutable

Una vez nuestro programa es ejecutable, escribimos en el LXTerminal el siguiente comando:

```
sudo crontab -e
```

Se nos abrirá un fichero en el que escribiremos al final (guardamos los cambios con Ctrl+X y confirmamos con "Y"):

```
@reboot Python home/pi/server.pi &
```

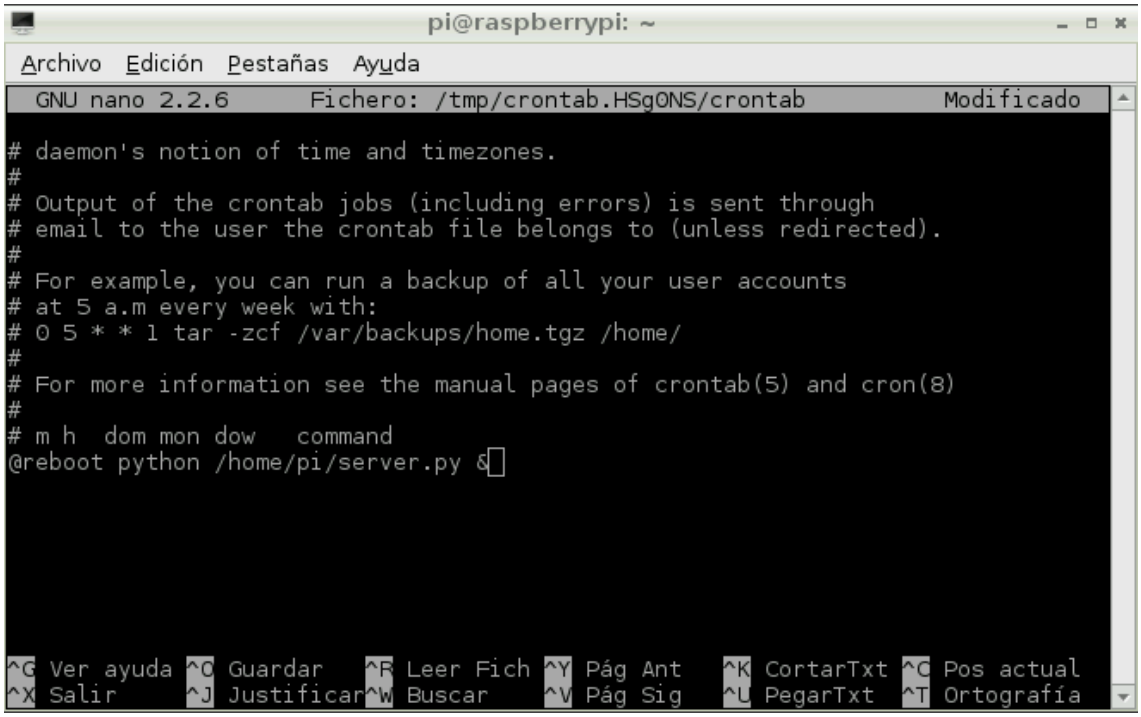


Ilustración 49 – Configuración de crontab (I)

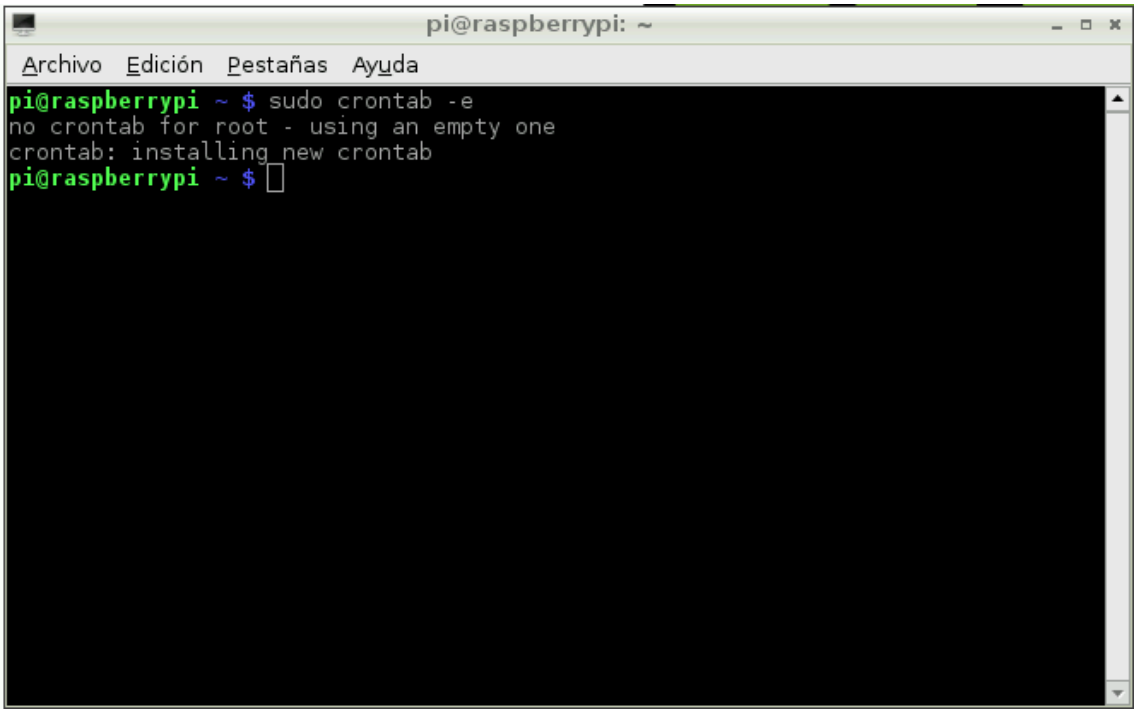


Ilustración 50- Configuración de crontab (II)

Una vez configurado, sólo tenemos que reiniciar la Raspberry Pi y el programa se ejecutará automáticamente, pudiendo así desconectar el teclado, ratón y pantalla.

7. Conclusiones

Como resultado final hemos conseguido una estación meteorológica para el telescopio robótico de bajo coste y con los resultados esperados. Aunque esta estación se ha diseñado para el uso en el telescopio robótico, debido a la facilidad con que se puede recabar información de ella desde cualquier ordenador usando un puerto TCP/IP, puede ser empleada en cualquier aplicación que requiera el conocimiento de las condiciones ambientales. Sólo se necesita crear un cliente que use el protocolo de mensajes desarrollado.

Lamentablemente debido a la cantidad de trabajo que esto conllevaba no se ha podido realizar la parte referida a la cámara para analizar el cielo por falta de tiempo, por lo que se ha decidido hacer una ampliación de este proyecto con dicha parte para el proyecto de fin de máster.

La experiencia ha sido muy grata, ya que he aprendido sobre infinidad de temas que no había visto hasta la fecha en el grado, además de lo interesante que es el manejo de hoy en día de aparatos como la Raspberry Pi, utilizados en todo tipo de campos de la ingeniería e informática como proyectos de bajo coste, además de la enseñanza.

7.1. Propuestas de mejora

Algunos de las posibles ampliaciones del proyecto son las siguientes:

- Crear una página web desde la que poder consultar en cualquier lugar los datos de la estación. Esto se llevaría a cabo mediante un cliente en un ordenador remoto que modifique una página html.

- Añadir un sensor de presión atmosférica.

- Colocar otro sensor de temperatura y humedad en el interior de la cúpula, para monitorizar las condiciones en las que se encuentran el telescopio y el resto de instrumentación alojada allí.

- Terminar el procesado de imágenes del cielo mediante la webcam con un objetivo de gran angular.

Anexo

MCP300X 10-Bit Analog-to-Digital Converters

Product Information

The Microchip MCP300X family of 10-bit analog-to-digital converters (ADCs) combines high performance and low power consumption in a small package – making it ideal for embedded control applications.

Consisting of the MCP3001, MCP3002, MCP3004 and MCP3008, the MCP300X family features a successive approximation register (SAR) architecture and an industry-standard SPI™ serial interface. Devices are available with 1, 2, 4 or 8 input channels and in PDIP, SOIC and TSSOP packages.

The MCP300X family offers existing Microchip customers added flexibility when incorporating analog inputs into their designs. The industry standard SPI interface allows 10-bit ADC capability to be added to any PICmicro® microcontroller. In addition, new customers will find the performance and price of the MCP300X family very attractive.

Applications for the MCP300X family include data acquisition, instrumentation and measurement, multi-channel data loggers, industrial PCs, motor control, robotics, industrial automation, smart sensors, portable instrumentation and home medical appliances.

Features:

- 200k samples/second
- 1, 2, 4 or 8 channels
- Low Power: 5 nA typical standby, 425 µA typical active
- ±1 LSB INL, ±1 LSB DNL
- No missing codes
- Industrial temperature range: -40°C to +85°C
- Single supply operation: 2.7V to 5.5V
- SPI serial interface
- PDIP and SOIC packages

Related Application Notes:

- AN679 Temperature Sensing Technologies
- AN684 Single Supply Temperature Sensing with Thermocouples
- AN685 Thermistors in Single Supply Temperature Sensing Circuits
- AN687 Precision Temperature Sensing with RTD Circuits
- AN699 Anti-Aliasing Analog Filters for Data Acquisition Systems

MCP300X 10-Bit Analog-to-Digital Converters *Continued*

Additional Information:

- Microchip's web site: www.microchip.com
- Microchip's *Technical Library CD-ROM*, Order No. DS00161
- More than 112 Application Notes available:
  – *Embedded Control Handbook,* Order No. DS00092
  – *Embedded Control Handbook, Volume 2, Math Library,* Order No. DS00167
- Microchip's *Overview, Quality Systems and Customer Interface System,* Order No. DS00169
- Third party software and hardware support:
  – Emulators
  – Programmers
  – Gang Programmers
  – Software Tools
  – Development Boards and Accessories
  – Design Consultants
  – *Third Party Guide,* Order No. DS00104

MCP300X High-Performance 10-Bit Analog-to-Digital Converters

Product	Resolution (Bits)	No. of Channels	Sampling Rate (ksps)	INL (±LSB)	DNL (±LSB)	Supply Voltage	Temperature Range	Standby Current @ 5V (typical, µA)	Operating Current @ 5V (typical, µA)	Packages
MCP3001	10	1	200	1	1	2.7 - 5.5	-40° to +85°C	0.005	400	8P, 8SO, 8TSSOP
MCP3002	10	2	200	1	1	2.7 - 5.5	-40° to +85°C	0.005	525	8P, 8SO, 8TSSOP
MCP3004	10	4	200	1	1	2.7 - 5.5	-40° to +85°C	0.005	425	14P, 14SO, 14TSSOP
MCP3008	10	8	200	1	1	2.7 - 5.5	-40° to +85°C	0.005	425	16P, 16SO

Development Tool Support

Microchip is offering a comprehensive set of support tools including application notes and the Analog Evaluation System. The evaluation system consists of the analog evaluation driver board, incorporating a PICmicro microcontroller, coupled with an MCP300X device-specific evaluation board. Windows® -based software features powerful data collection and analysis

Americas

Atlanta	(770) 640-0034
Austin-Analog	(512) 345-2030
Boston	(978) 692-3848
Boston-Analog	(978) 371-6400
Chicago	(630) 285-0071
Dallas	(972) 818-7423
Dayton	(937) 291-1654
Detroit	(248) 538-2250
Los Angeles	(949) 263-1888
Mountain View-Analog	(650) 968-9241
New York	(631) 273-5305
San Jose	(408) 436-7950
Toronto	(905) 673-0699

Asia/Pacific

Australia	61 2 9868 6733
China-Beijing	86 10 85282100
China-Shanghai	86 21 6275 5700
Hong Kong	852 2401 1200
India	91 80 2290061
Japan	81 45 471 6166
Korea	82 2 554 7200
Singapore	65 334 8870
Taiwan	886 2 2717 7175

Europe

Denmark	45 4420 9895
France	33 1 69 53 63 20
Germany	49 89 627 144 0
Germany-Analog	49 89 895650 0
Italy	39 039 65791 1
United Kingdom	44 118 921 5869

As of 02/01/01

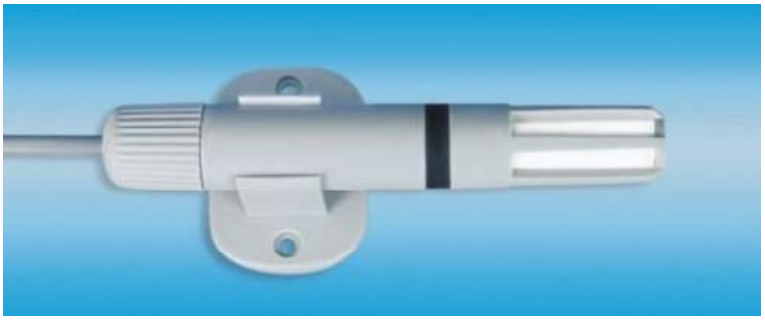
MICROCHIP
The Embedded Control Solutions Company®

Microchip Technology Inc. • 2355 W. Chandler Blvd. • Chandler, AZ 85224-6199 • (480) 792-7200 • Fax (480) 792-9210

AOSONG

Digital temperature and humidity sensor

AM2315 Product Manual

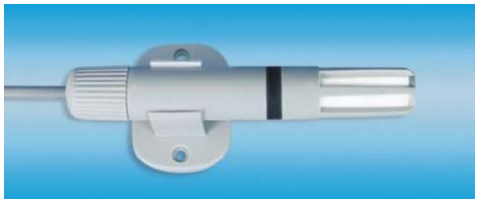


www.aosong.com

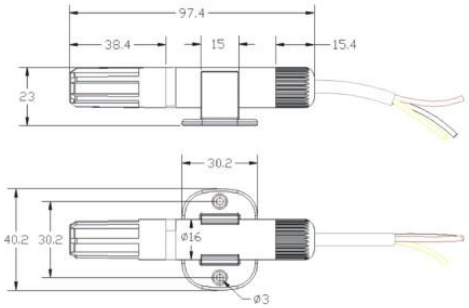
1、Product Overview

AM2315 capacitive humidity sensing digital temperature and humidity sensor is one that contains the temperature and humidity combined sensor has been calibrated digital signal output. Special temperature and humidity acquisition technology, to ensure that the product has high reliability and excellent long-term stability. The sensor includes a capacitive sensor wet components and an integrated high-precision temperature measurement devices, and connected with a high-performance microprocessor. The product has excellent quality, fast response, strong anti-jamming capability, and high cost.

AM2315 communication standard I2C communication. I2C communication standard communication timing, the user can be directly linked to on the I2C communication bus, no additional wiring, simple to use. Transmission of digital data output directly compensated by the temperature, humidity, temperature and Check (CRC) and other digital information, the user does not need the digital output for the second calculation, and no need for temperature compensation of the humidity, you can get an accurate temperature and humidity information . Product of four leads, convenient connection, special packages according to user needs.



Physical map



Dimensions (unit: mm)

2、Applications

HVAC air conditioners, dehumidifiers, testing and inspection equipment, consumer goods, automotive, automation, data loggers, weather stations, home appliances, humidifiers, medical, and other related humidity measurement and control.

3、Features

Completely interchangeable, low cost, long-term stability, relative humidity and temperature measurement, long distance signal transmission, digital signal output, precise calibration, low power consumption, standard I^2C bus digital interface.

4、Interface definition

4.1 AM2315 pin assignment

Table 1: AM2315 pin assignment

Pin	Color	Name	Description
1	Red	VDD	Power (3.5V-5.5V)
2	Yellow	SDA	Serial data, bidirectional
3	Black	GND	Ground
4	White	SCL	Serial Clock, input

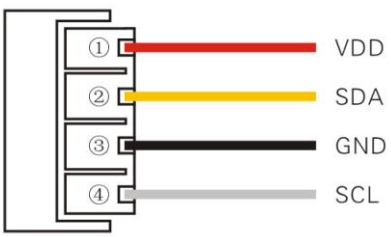


Figure 1: AM2315wiring diagram

4.2 Power supply pins（VDD GND）

AM2315 supply voltage range of 3.5 - 5.5V, recommended supply voltage is 5V.

4.3 Data line（SDA SCL）

SCL I²C communication signal line when the clock line, SCL used for communication between the microprocessor and the AM2315 synchronous. SDA pin is tri-state structure for reading, writing sensor data. Specific communication timing, see a detailed description of communication protocols.

5、Sensor Performance

5.1 Relative Humidity

Table 2: AM2315 relative humidity performance table

Parameter	Condition	min	typ	max	Unit
Resolution			0.1		%RH
			16		bit
Accuracy [1]	25℃		±2		%RH
Repeatability			±0.1		%RH
Exchange		Completely interchangeable			
Response [2]	1/e(63%)		<5		S
Sluggish			<0.3		%RH
Drift [3]	Typical		<0.5		%RH/yr

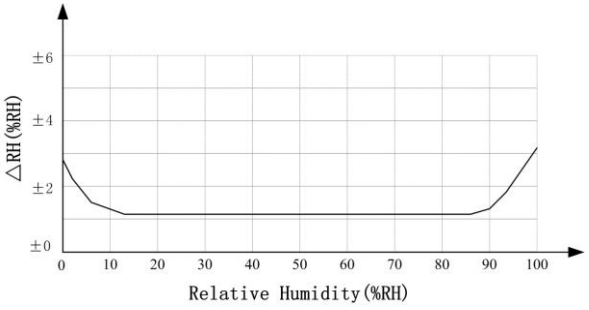


Figure 2: 25 ℃ relative humidity when the AM2315 maximum error

5.2 Temperature

Table 3: AM2315 relative temperature performance table

Parameter	Condition	min	typ	max	Unit
Resolution			0.1		℃
			16		bit
Accuracy			±0.1	±1	℃
Range		−40		125	℃
Repeat			±0.2		℃
Exchange		Completely interchangeable			
Response	1/e(63%)		<5		S
Drift			±0.1		℃/yr

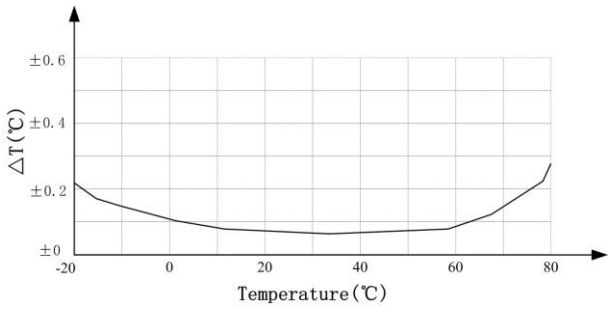


Figure 3: The maximum temperature error of temperature sensor

6、Electrical Characteristics

Electrical characteristics, such as energy consumption, high, low, input, output voltage, depends on power. Table 4 details the AM2315 electrical characteristics, if not stated otherwise supply voltage of 5V. For best results with the sensors, design strictly in accordance with Table 4 and Table 6 of the conditions of the design.

Table 4: AM2315 DC Characteristics。

Parameter	Condition	min	typ	max	Unit
Supply voltage		3.5	5	5.5	V
Power [4]	Dormancy	10	15		μA
	Measuring		500		μA
	Average		300		μA
Low-level output	I_{OL}[5]	0		300	mV
High output	Rp<25 kΩ	90%		100%	VDD
Low input voltage	Decline	0		30%	VDD
Input High	Rise	70%		100%	VDD
Rpu[6]	VDD = 5V VIN = VSS	30	45	60	kΩ
Output current	On		8		mA
	Off	10	20		μA
Sampling period		2			S

[1] The accuracy of the factory test, the sensor at 25 ℃ and 5V, the accuracy of indicators under the conditions tested, it does not include hysteresis and non-linear, and only for non-condensing environment.

[2] at 25 ℃ and 1m / s air flow conditions, to reach 63% of first-order response time required.

[3] in volatile organic compounds, the values may be higher. See manual application to store information.

[4] This value is VDD = 5.0V, when the temperature is 25 ℃, 2S / time, average conditions.

[5] low output current.

[6] that pull-up resistor.

7、I²C Communication protocol

7.1 I²C communication protocol introduced

7.1.1 I²C About the bus

AM2315 and micro-processor control interface form of the I²C serial bus, in this brief introduction about the I²C bus protocol standard. Due to space limitations, the agreement can not list the entire contents of a deeper problem, please refer to the relevant information (refer to the Philips site inspection).

7.1.2 I²C Bus Overview

Philips (Philips) at 20 years ago invented a simple two-way two-wire serial communication bus, the bus is known as the Inter-I²C bus. I²C bus has now become the industry standard solution for embedded applications, is widely used in variety of microcontroller-based professional, consumer and telecommunications products, as a control, diagnostics and power management bus. More in line with standard I²C bus devices can communicate with an I²C bus, without the need to address decoder.

I²C bus only composed by the two signal lines, a serial data line SDA, the other root is the serial clock line SCL. I²C bus devices generally have their SDA and SCL pins are open-drain (or open collector) output structure. Therefore, the actual use, SDA and SCL signal lines must be pull-up resistor (Rp, Pull-Up Resistor). Pull-up resistor on the general value of 3 ~ 10 kΩ . Therefore, when the bus is free, the two signal lines remain high, almost no current consumption; electrical compatibility, and supports a variety of different voltage logic interface; different between the two can be directly connected to the bus, not require additional conversion circuitry to support a variety of ways a master multi-slave communication is the most common means of communication. It also supports dual-host communications, multi-host communications, and broadcast mode, and so on.

I²C typical configuration is shown in Figure 4。

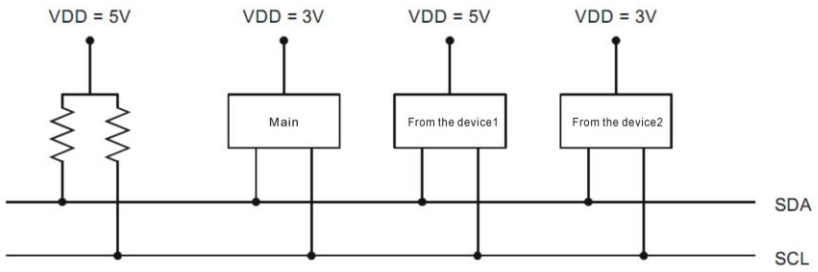


Figure 4: I²C typical configuration

7.1.3 I²C Bus protocol specification

◎ I²C Bus Definition of Terms

I²C–bus serial data (SDA) and serial clock (SCL) line connected to the bus, allowing the device to transmit information between each device has a unique address identification, and can be used as a transmitter or receiver (the device features decision), the device performs data transmission can also be seen as a master or a slave, the host is initialized bus allows data transfer and generates the clock signal transmission device. At this point, any device addressed is considered a slave. Details of the I²C bus definition of terms in Table 5.

◎ I²C Bus transfer rate

I²C Bus communication speed controlled by the host, to fast to slow. However, the maximum rate is limited, I²C bus data transfer rate up to the standard mode, fast 100Kb /s.

◎ I²C Bus bit transmission

I²C Bus bit transmission is through the data line SDA and SCL line to complete the two lines together. In the high period of SCL clock line, data line SDA low logic level, said current transmission "0"; in the high period of SCL clock line, data line SDA high logic level, said current transmission "1." Logic "0" (low) and "1" (high) level, is related by the VDD voltage level (for details see Table 4 AM2315 DC Characteristics table). In addition, each data bit transferred on to generate a clock pulse.

Table 5: I²C Bus definitions of terms

Term	Description
Transmitter	Devices to send data to the bus
Receiver	Device receiving data from the bus
Host	Initialization and termination of the clock signal sent sent produce devices
From machine	Addressed by the host device
Multi–host	At the same time try to control more than one host, but does not destroy the message bus.
Arbitration	It's a multiple hosts simultaneously try to control the bus but only one is allowed to control the bus and make the message is not destroyed in the process.
Synchronous	Two or more devices synchronized clock signal in the process

◎ **The validity of data**

Data line SDA of the data must be in the HIGH period of the clock remain stable. SDA data line high or low state only in the clock line SCL low time was allowed to change. But in the beginning and end of the I^2C bus when the exception (for details see Start and Stop conditions). Some other data may require the serial bus clock signal edge (rising or falling edge) is valid, but the level I^2C bus is valid. Specific timing diagram shown in Figure 5.

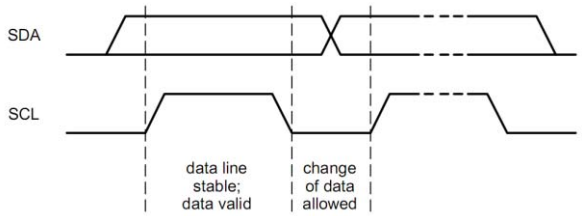


Figure 5: I^2CBus bit transmission

◎ **Start and stop conditions**

Initial conditions: During the time when SCL is high, SDA from high to low transition generated when the initial conditions. Produced in the initial conditions after the bus is considered busy. Initial conditions that are often denoted by S.

Stop condition: During the time when SCL is high, SDA from low to high transition when a Stop condition. Generate a stop condition after the bus is idle. Stop condition denoted by P.

Start and stop conditions diagram shown in Figure 6。

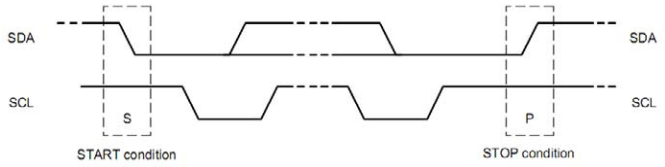


Figure 6: Schematic diagram of the start and stop conditions

◎ **Byte transfer format**

I^2C Bus to send and receive data bytes. Transferred to the SDA line must be 8−bit per byte. The number of bytes per transfer is unrestricted. First, the data transmission is the highest bit (MSB No. 7), the last transmission is the lowest bit (LSB, bit 0). In addition, each byte must be followed by an acknowledge bit (ACK). I^2C transmit data in Figure 7.

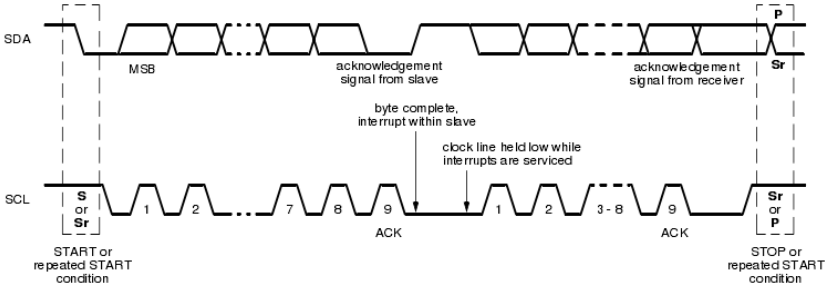


Figure 7: I^2C Bus data transfer

◎ I²C Bus response

I²C–bus transfers data in the process, each transmission of a byte, must be answered with a status bit. The case of the receiver can receive data acknowledge bit to inform the transmitter. Acknowledge bit, still the master generates the clock pulse, and an acknowledge bit, the data state is follow the "who who receives a" principle, which always generates the acknowledge bit by the receiver, in response to the clock pulse period the receiver must pull the SDA line low, making it the clock pulse is stable LOW during the HIGH period (see Figure 8), of course, setup and hold times must be considered (for details see table 6). Send data from master to slave, the slave generates the acknowledge bit; host to receive data from the slave, the acknowledge bit by the master.

I²C bus standard: 0 acknowledge bit to the receiver acknowledge (ACK), often abbreviated as A; of 1 indicates non–response (NACK), often abbreviated to NA. After the transmitter sends the LSB should release the SDA line (pulled SDA), to wait for the receiver generating an acknowledge bit.

If the receiver is receiving last byte of data, or can not receive more data, it should produce non–ACK to notify the sender. If you find the receiver transmitter generates a non–response state, you should terminate the transmission.

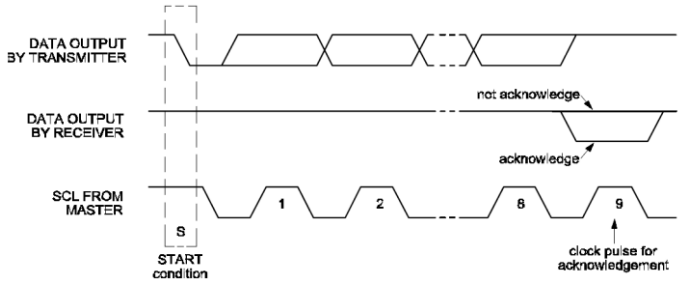


Figure 8: I²CBus response

◎ Slave Address

No additional I²C bus address decoder and chip select signals. Multiple devices with I²C bus interface can be connected to the same I²C bus, and address them through the device to distinguish between. The process of addressing the I2C bus is usually the initial conditions in the first byte after the decision to choose which one from the host machine, that is 7–bit addressing address (the other is 10–bit addressing address is different, the sensors 7–bit addressing address). The first byte of the bit definitions shown in Figure 9, the first byte of the first seven bits of the slave address, the lowest bit (LSB) is No. 8. It determines the direction of the message, the first byte of the least significant bit (LSB) is "0": Indicates that the host will write the information to be selected from the machine; "1" indicates that the master will read information from the machine.

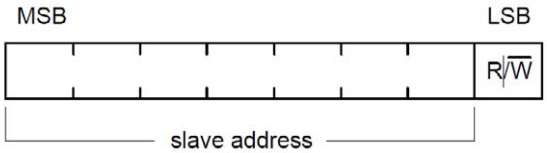


Figure 9: After the initial conditions of the first byte

Sent an address, the system of each device in the initial conditions, the first seven and more if its own address as the device will think it is the host address, as a slave receiver or slave transmitter by the R / W bits. Host is the master Parts, it does not need the device device address, and other devices are all from the machine, have the device address. Must ensure that all on the same I²C bus slave addresses are uniquely determined, can be repeated, or the I²C bus will not work.

◎ Schematic diagram of the basic data transfer format

Figure 10 and Figure 11 are given the I²C data transmit and receive the basic format. It should be noted, Figure 11 and Figure 12 is different, in Figure 11, the host sends to the last byte from the data, the response from the machine may or may not answer, but in any case the host can generate stop condition. If the host sends data to the slave (even including slave address) when the detected non-response from the machine, you should promptly stop the transmission.

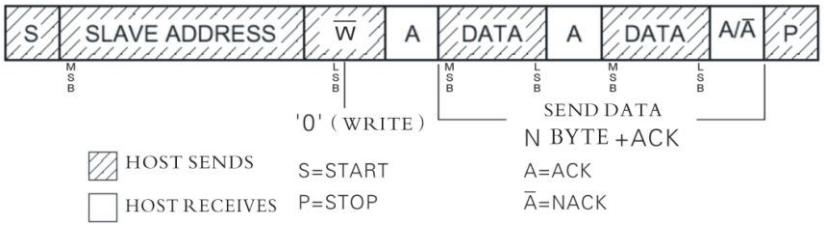


Figure 10： I²C The bus master sends data to the basic format from

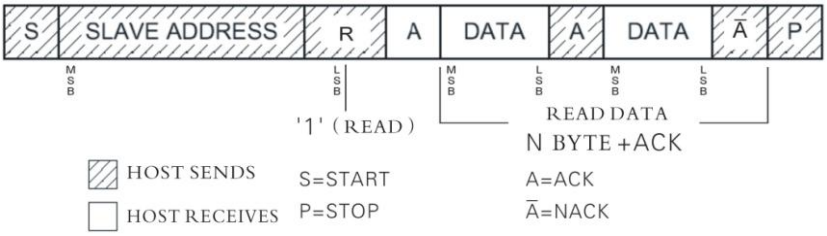


Figure 11： I²CThe bus master receives data from the slave's basic format

7.2 AM2315 Sensor I²C communication protocol

AM2315 I²C bus serial interface, in full accordance with the standard I2C protocol addressing, can be directly linked to the I²C bus. AM2315 sensor I2C address (SLAVE ADDRESS) is 0xB8, the I²C bus protocol based on the standard, based on ModBus protocol, developed a unique communication protocol, reducing the transmission error rate. Microcontroller read AM2315 sensor, the sensor I²C_ModBus in strict accordance with AM2315 communication protocol and timing design.

7.2.1 I²C Interface Description

AM2315 digital temperature and humidity sensor as a slave, and the host (user microprocessor) way communication between the I²C bus standard mode. For the accurate measurement of humidity, temperature and humidity to reduce the impact, AM2315 sensors in the non-working period, automatically be converted to sleep, to reduce the work consumption, to reduce the sensor self-heating of the humidity of the surrounding environment. AM2315 uses passive mode, that is, the host through the instruction wake-up sensor, and then sends the appropriate commands, read the corresponding values of temperature and humidity; communication after the acquisition of temperature and humidity sensor is triggered once; so if the long did not read the sensor, please read the two consecutive second sensor (minimum interval of two to read 2S), the second is the latest measured value; collected after the end of the sensor automatically be converted to sleep. Next host to be read sensor, the need to re-awaken the sensor. Must be noted that host communication from start to finish, for a maximum of 3S. If communication is not completed within 3S, sensors automatically end communication, automatically be converted to sleep, read again the host, such as sensors, need to re-send the wake-up command.

7.2.2 I²C Interface Features

This section describes the AM2315 Sensor I²C interface characteristics, if you want to get the best communications with the sensor results, designed strictly in accordance with Figure 12 and Table 6 of condition design.。

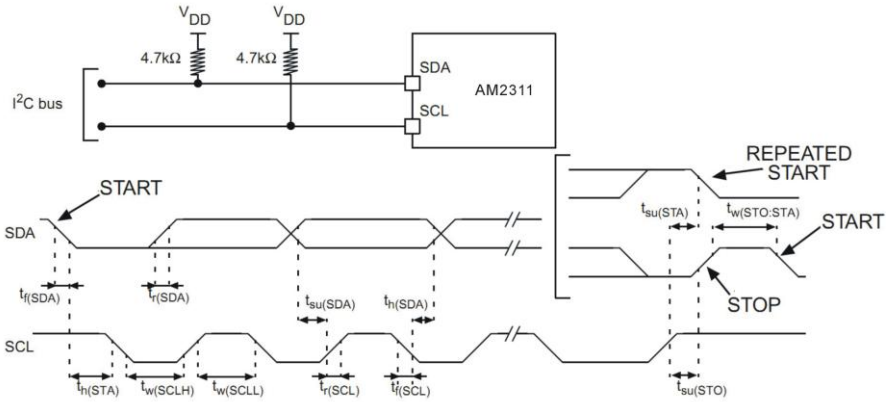


Figure 12： AM2315 Typical application circuit and the I2C bus timing diagram

Table 6： AM2315 Sensor I²C interface characteristics

Symbol	Parameters	Standard I²C mode		Unit
		min	max	
SCLClock frequency			100	kHz
tw(SCLL)	SCLClock low time	4.7		μs
tw(SCLH)	SCLClock high time	4.0		
tsu(SDA)	SDA Setup time	250		ns
th(SDA)	SDA Data hold time	0[1]		
tr(SDA) tr(SCL)	SDA & SCL Rise time		1000	
tf(SDA) tf(SCL)	SDA & SCL Fall Time		300	
th(STA)	Start condition hold time	4.0		μs
tsu(STA)	Repeated START condition setup time	4.7		
tsu(STO)	Stop condition setup time	4.0		μs
tw(STO:STA)	Stop to Start condition time (bus free)	4.7		μs
Cb	Capacitive load for each bus		400	pF

7.2.3 Communication protocol

AM2315 Sensor I²C communication protocol is a standard I²C bus protocol based on the reference ModBus protocol, the sensor according to AM2315 own characteristics, a combination of I²C_ModBus agreement. The following format:：

◎ Communication data (information frame) format

Data format：

Data length：

I^2CAdd+R/W	Function code	Data area	CRC Check [3]
1Byte [2]	1Byte	NByte	16-bit CRC (cycle redundancy code)

[1] If the interface does not allow to extend the time low, only need to comply with the longest hold time Start condition.

[2] A byte consists of 8-bit binary number (both 8 bit).

[3] CRC checksum algorithm, for details see: CRC code calculation method; detailed below.

◎ Communication and information transfer process

When the communication command from the sending device (host) to the sensor, I^2C address of the command line with the sensor, the sensor was to receive, and in accordance with the relevant requirements of the function code and read the information; and the implementation of the results (data) back to the host. The information returned includes the function code, data and after the implementation of the CRC code (user time to read the CRC, stop conditions can be sent directly).

◎ Communication I^2C slave address

AM2315 I^2C addresses are the same for each sensor, and is 0xB8. Therefore, in the same bus can only be linked to a AM2315 sensor, the sensor only after receiving the start signal and the same with its I^2C address only to respond to the host.

◎ I^2C communication function code

Communication of information for each function code is the first byte of the frame transmission. I^2C_ModBus communication rules, define the function code is 1 to 127. As a host request, through the function code tells the slave what action should be implemented. Response as a slave, the slave returns the function code and the functions from the host to send the same code, it indicates that the response from the host machine and have been associated operations. I^2C_ModBus part of the function codes described in Table 7.

Table 7：I^2C_ModBus Part of the function code

Function code	Definition	Operations (binary)
0x03	Read register data	Read data from one or more registers
0x10	Write multiple registers	Multiple sets of binary data is written to multiple registers

◎ I^2C communication data area

Data area including the need to return to what information from the sensor or perform any action. This information can be data (such as: temperature, humidity, the sensor device information, user-written data, etc.), the reference address. For example, the host told the sensor 03 through the function code value return register (read register contains the starting address and length of register read), the returned data includes the length of the data register and data register contents.

I²C_Modbus sensor uses a custom communication protocol, host, use of communication commands (function code 03), can read the data register any of its data register table shown in Table 8. Sensor temperature and humidity data register stores the value and the corresponding sensor signal equipment and other related information; each data register is a single byte (8 bits) of binary data; read sensor, up to 10 registers of data, more than reading take the length of the sensor will return the corresponding error code. Error code information, see Table 1.

Table 8: AM2315 Data register table

Register information	Add	Register information	Add	Register information	Add	Register information	Add
High RH	0x00	Model high	0x08	Users register a high	0x10	Retention	0x18
Low RH	0x01	Model low	0x09	Users register a low	0x11	Retention	0x19
High temp.	0x02	Version number	0x0A	Users register 2 high	0x12	Retention	0x1A
Low temp.	0x03	ID(24-31) Bit	0x0B	Users register 2 low	0x13	Retention	0x1B
Retention	0x04	ID(16-23) Bit	0x0C	Retention	0x14	Retention	0x1C
Retention	0x05	ID(8 - 15) Bit	0x0D	Retention	0x15	Retention	0x1D
Retention	0x06	ID(0 - 7) Bit	0x0E	Retention	0x16	Retention	0x1E
Retention	0x07	Status Register	0x0F	Retention	0x17	Retention	0x1F

◎ **Temperature output format**

Temperature resolution is 16Bit, the maximum temperature position (Bit15) equal to 1 indicates a negative temperature, the temperature highest (Bit15) is equal to 0 for positive temperature; temperature in addition to highest (Bit14 ~ Bit0) that string out of the temperature sensor. String out of the temperature sensor is 10 times the actual temperature;

◎ **Status Register**

Status register, Bit7-Bit0 bit, temporarily reserved

Status register bits	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Features	Retention							

◎ **I²C_ModBus Function Code Description**

一、**Function code "03":** Read multiple sensors register

Host sends a read frame format:

START+ (I²CAddress+W)+ Function code (0x03) + Start address + Register number +STOP

The host to read back the data:

START+ (I²C Add+R)+ Continuously read sensor data returned +STOP

Sensor response frame format:

Function code (0x03) + Register number + Data +CRC[1]

for example: The host continues to read sensor data: 0x00 start address register of the four sensor data.

Sensor data register address and data:

Register add	Register data	Data on	Register add	Register data	Data on
0x00	0x01	High RH	0x02	0x00	High temp.
0x01	0xF4	Low RH	0x03	0xFA	Low temp.

Host sends the message format:

Host sends	Bytes	message	Remarks
Sensor add	1	0xB8	Sensor I²C address（0xB8）+W（0）
Function code	1	0x03	Read the register
Start add	1	0x00	Register start add 0x00
Register No.	1	0x04	Read the register number

Sensor response to the return message format:

Response from machine	Bytes	Send	Remarks
Function code	1	0x03	Read the register
Returns No. of bytes	1	0x04	Back 4 of 4 bytes register
Register 1	1	0x01	Add 0x00 of the content (RH high byte)
Register 2	1	0xF4	Add 0x01 of the content (RH low byte)
Register 3	1	0x00	Add the content of 0x02 (temp. high byte)
寄存器 4	1	0xFA	Add 0x03 of the content (temp. low byte)
CRC 码	2	31A5	Back to the CRC, the low byte first ;

[1]: Details of the CRC calculations see the back of the CRC introduced, the sensor returns all the data are CRC check, the user can choose to read or not read。

Numerical:

Read back from the sensor temperature and humidity values, as long as the value into a decimal number 10 is divided by the value corresponding to the temperature and humidity, the temperature of the corresponding unit ℃, humidity units of % RH. For example, read back the data above:

Humidity: 01F4 = 1×256+15×16+4 = 500 => Humidity = 500÷10=50.0%RH;

Temp.: 00FA= 15×16+10 = 250 => Temp. = 250÷10 = 25.0℃

➘ **Note:** CRC verification code CRC is calculated by the code to arrive, and then pass the CRC and the sensor compared; the same, the data is received correctly, or that the data has errors.

二、 **Function code "10":** write multiple registers to sensor

The host can use this function code multiple data stored in the register to the sensor. AM2315 sensors register a single one-byte or 8 bits. Sensors allow you to save a maximum of 10

data registers. Therefore, the host single sensor to save up to 10 registers. More than 10, the sensor will return the corresponding error code.

Host sends write frame format:

START+(I²C Add+W)+ Function code (0x10)+ Start address + Register number + Save the data +CRC+STOP

Confirm the host to read instructions:

START+(I²C Add+R)+ Read the sensor data returned +STOP

Sensor response frame format:

Function code（0x10）+ Start Add + Register number +CRC

For example: Should save the address of the host 01, 02 10, 11 of the sensor to register.

Host sends the message format:

Host sends	Bytes	Send	Remarks
Sensor Add	1	0xB8	Sensor I²C Add （0xB8）+W（0）
Function code	1	0x10	Write multiple registers
Start Add	1	0x10	To write the start address register
Byte length	1	0x02	Save the data word length (2 Byte)
Data 1	1	0x01	Save the data (address: 10)
Data 2	1	0x02	Save the data (address: 11)
CRC Code	2	C092	Host calculated CRC code, low byte first (I²C addresses not included in the CRC calculation)

Sensor response to the return message format:

Response from machine	Bytes	Send message	Remarks
Function Code	1	0x10	Write multiple registers
Start address	1	0x10	Save the start address
Save the data length	1	0x02	Sensors save data length
CRC Code	2	FC04	Sensors return the CRC calculation, the low byte first ;

◎ **CRC Check**

Host or check code can be used to discriminate the sensor to receive information is correct. As the electronic noise or some other interference, the information during transmission error sometimes occurs, the error check code (CRC) can verify that the host or sensor data transmission process in the communication of information is wrong, wrong data can be discarded (either send or receive), which increases the system's security and efficiency.

I²C_ModBus protocol of CRC (cycle redundancy code) consists of 2 bytes, or 16-bit binary number. CRC code from the sending device (host), is placed in the rear send a message frame, I²C address is not included in the CRC calculation. CRC are either sent or received by the low byte first, high byte format after sending.

Receive information (sensors) and then recalculate the information received by the CRC, the CRC is calculated by comparing the receiver to match, if they do not match, then the error. Users need special

Do note that the sensor reading instruction without adding the CRC; write sensor, the CRC must increase; and all the return data are CRC.

◎ **CRC code is calculated**

1. Preset a 16−bit registers as hexadecimal FFFF (that is, all 1); call this register is the CRC register;
2. The first 8−bit binary data (communication of information both the first byte of the frame) and 16−bit CRC register, or the lower 8 bits are different, the results put in the CRC register;
3. The CRC register right one (towards low) with 0 fill the highest place, and check out right after the bit;
4. If out of the bit is 0: Repeat Step 3 (again shifted to the right one); if out of place as 1: CRC register with the polynomial A001 (1010 0000 0000 0001) XOR;
5. Repeat steps 3 and 4, until the right 8 times, so that all of the 8−bit data were processed;
6. Repeat steps 2 through 5, the communication frame to the next byte of information processing;
7. All bytes of the communication of information frames calculated according to the above steps completed, the resulting 16−bit CRC register is the high and low bytes are exchanged;
8. Finally, get the contents of the CRC register is: CRC code.

◎ **CRC code calculation code in C language**

Description: This program calculates the length of * ptr within the first len bytes of the CRC.

```
unsigned short crc16(unsigned char *ptr, unsigned char len)
{
    unsigned short crc=0xFFFF;
    unsigned char i;
    while(len--)
    {
        crc ^=*ptr++;
        for(i=0;i<8;i++)
        {
            if(crc & 0x01)
            {
                crc>>=1;
                crc^=0xA001;
             }else
             {
                crc>>=1;
             }
        }
    }
    return crc;
}
```

7.2.4 I²C Communication Timing

AM2315 sensor I²C communication, although communication is based on standard I²C timing, but

necessary according to our protocol and communication timing requirements, in order to accurately read the sensor. Please strictly in accordance with protocol design and timing for reading.

◎ I²C Read the complete sequence example

Figure 13 shows a complete example of the sensor to read and write and read and write special time requirements, in strict accordance with specific time requirements to read and write, or they will not read the sensor or data appear incorrect and so on. Timing diagram of several special needs attention, detailed figure of the time requirements; host communication from start to finish, for a maximum of 3S.

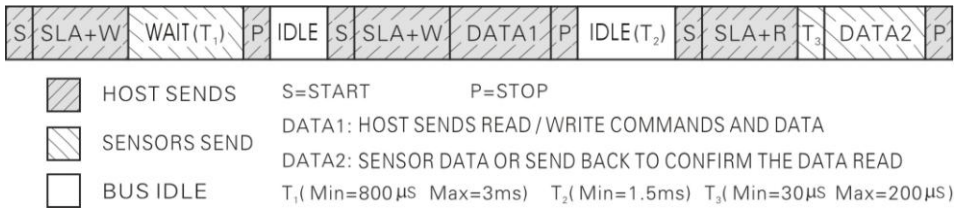


Figure 13： I²C Sensor reading and writing a complete sample chart

◎ I²C Read and write timing decomposition

Read or write the sensor, the following three steps must be, otherwise it will not communicate or can not read the correct data：

Step one： Wake-up sensor

In order to reduce the humidity sensor self-heating caused the error, the sensor in the non-working state, in a dormant state, so the sensor must wake up to read the sensor, in order to send read and write commands, otherwise the sensor will not respond. It should be noted that, in the wake sensor, sending the I2C address, the sensor will not respond to ACK, but the host must send ACK back the clock to confirm that the ninth SCL clock signal. Wake up the sensor's operating instructions as follows:

Sends a start signal is applied to host the start address, wait for a period of time (to wait for at least 800μ s, the largest 3ms; if the host is a hardware I2C, you do not need to wait, to wait for hardware I2C automatically), then sends a stop signal.

Namely: the initial signal +0 xB8 + wait (800us-3ms) + stop signal timing diagram shown in Figure 14.

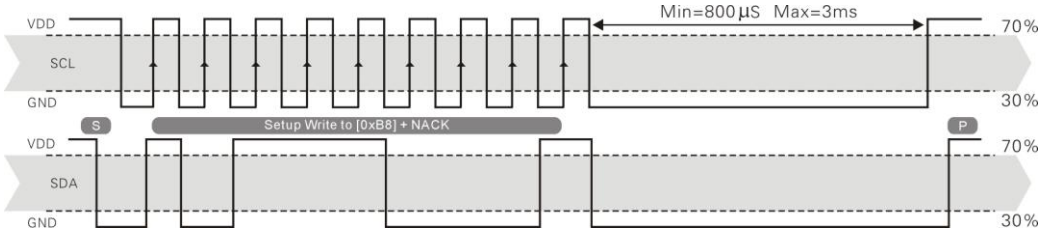


Figure 14: Wake-up sensor

Step Two read commands to send or send written instructions

Wake AM2315 sensors, can be read in full accordance with standard I²C timing, the maximum speed supported 100Kb / s. Read temperature and humidity sample, shown in Figure 15.

Host to send commands to: START +0 xB8 (SLA) +0 x03 (function code) +0 x00 (start address) +0 x04

(register length) + STOP

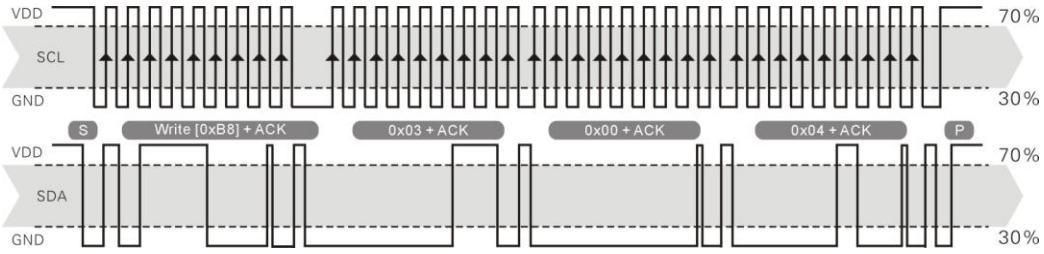


Figure 15: Sends a read command example temperature and humidity

Step three data read back or confirmation signal

Send read / write command, the host must wait at least 1.5ms, and then send a read sequence, to read back the data example shown in Figure 16; to note is that you read the data, finished his I2C address is required wait at least 30μ s before sending over the next serial clock, the read data, or communication error will occur.

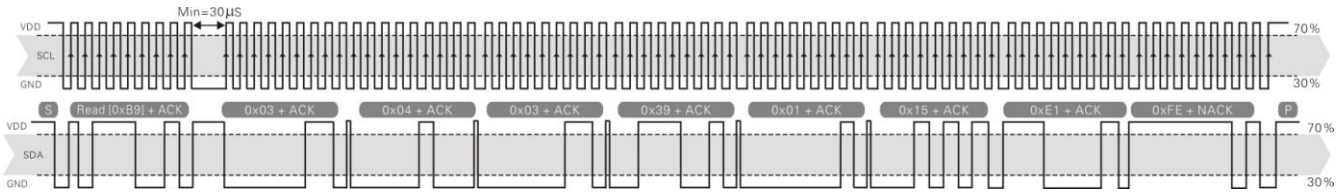


Figure 16: Example values of temperature and humidity reading

Data is read back to the host:

0x03(Function code)+0x04(Data length)+0x03(High humidity)+0x39(Low humidity)+

0x01(High temperature)+0x15(Low temperature)+0xE1(CRC Low byte checksum)+ 0xFE(CRC Checksum high byte);

So: 0339H = 3×256+3×16+9 = 825 => Humidity = 825÷10=82.5%RH;

0115H = 1×256+1×16+5 = 277 => Temperature = 277÷10=27.7℃

Through the above three steps to complete all the registers of the sensor reading and writing operations (users can write registers, only five, namely, the status register, four user registers at the same time, the status register, can be written separately, otherwise an error); users in the design, follow these three steps must be fully read and write.

Sensors send the data, trigger a temperature and humidity measurement; measurement is completed, record temperature and humidity values, then a communication completed, the sensor automatically into hibernation; so long as the sensor does not read, read the second consecutive sensors, back to the second reading of the temperature and humidity for the latest value (minimum interval of continuous reading 2S).

7.2.5 Peripherals read the flow chart

AM2315 schematic diagram I2C sensor reading is shown in Figure 17, while our company also provides sample code to read C51, customers need to download, please visit our website (www.aosong.com) associated download, this manual does not provide code instructions.

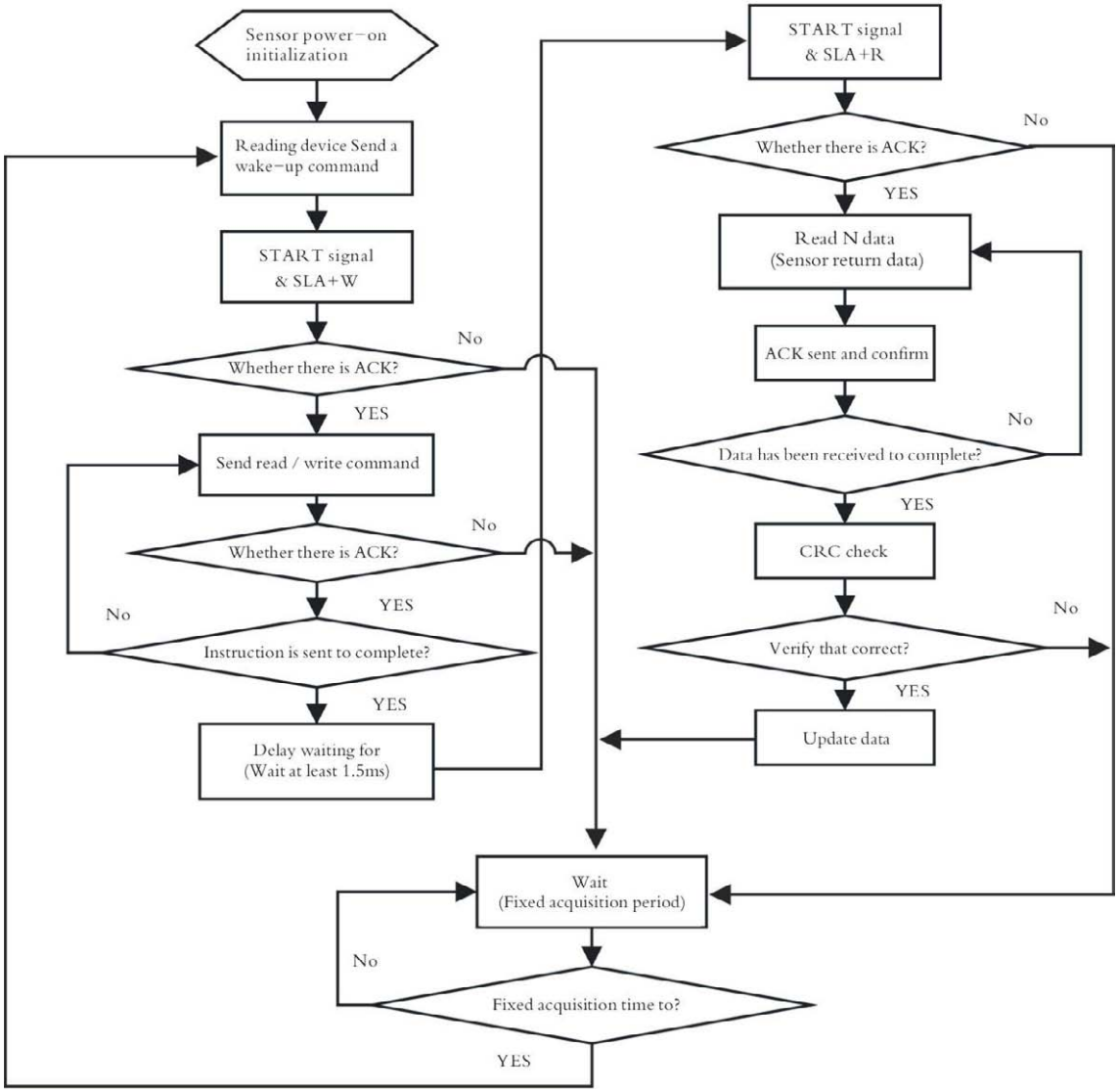


Figure 17: Sensor with I2C read flow chart

Table 1: I²C_MODBUS communication protocol summary table

Read the bus Description: I2C address is 0xB8; access a maximum of 10 registers;

Read the bus communication time for a maximum of a 3 S。

Each returned sensor data plus the CRC, the user can choose not to read the CRC。

Sensor reading frame format: Host frame format: (SLA+W)+ Function code (0x03) + Start Add + Register number

From the machine frame format: Function code (0x03) + Data length + Return data +CRC

Write frame format sensor: Host frame format: (SLA+W)+ Function code (0x10) + Start Add + Register number + Save the data +CRC

From the machine frame format: Function code (0x03) + Start Add + Register number +CRC

AM2315 sensor registers list:

Register information	Add	Register information	Add	Register information	Add	Register information	Add
High RH	0x00	Model high	0x08	Users register 1 high	0x10	Retention	0x18
Low RH	0x01	Model low	0x09	Users register 1 low	0x11	Retention	0x19
High temp.	0x02	Version number	0x0A	Users register 2 high	0x12	Retention	0x1A
Low temp.	0x03	ID(24−31) Bit	0x0B	Users register 2 low	0x13	Retention	0x1B
Retention	0x04	ID(16−23) Bit	0x0C	Retention	0x14	Retention	0x1C
Retention	0x05	ID(8 − 15) Bit	0x0D	Retention	0x15	Retention	0x1D
Retention	0x06	ID(0 − 7) Bit	0x0E	Retention	0x16	Retention	0x1E
Retention	0x07	Status Register	0x0F	Retention	0x17	Retention	0x1F

Status Register Definition: Bit7−Bit0 bit reserved;

Temperature format: temperature highest (Bit15) equal to 1 indicates a negative temperature, the temperature highest (Bit15) is equal to 0 for positive temperature; temperature in addition to the highest bit (Bit14 ~ Bit0) that string out of the temperature sensor. String out of the temperature sensor is 10 times the actual humidity value;

Write sensor: Register is available for users to write (0x0F ~ 0x13); others prohibit write register and status register can only be written separately。

Reading and writing sample:

Function	code	Start Add	Frame data content
Read temperature and humidity	0x03	0x00	Send：(SLA+W)+0x03+0x00+0x04
			Return：0x03+0x04+ High RH + Low RH + High temp. + Low temp. +CRC
Read the temp.	0x03	0x02	Send：(SLA+W)+0x03+0x02+0x02
			Return：0x03+0x02+ High temp. + Low temp. +CRC
Humidity reading	0x03	0x00	Send：(SLA+W)+0x03+0x00+0x02
			Return：0x03+0x02+ Low RH+ Low RH+ CRC
Read device information	0x03	0x08	Send：(SLA+W)+0x03+0x08+0x07
			Return：0x03+0x07+ Model (16)+ Version (8)+ ID(32)+CRC
Write Status Register	0x10	0x0F	Send：(SLA+W)+0x10+0x0F+0x01+0x01+0xF4 (Low) +0xB7 (High) Note: Function code + register start address register number + Save + content + CRC
			Return：0x10+0x0F+0x01+0xB4 (Low byte) +0x35 (High byte) Note：Function code + register start address + register number + CRC
Write user registers 1	0x10	0x10	Send：(SLA+W)+0x10+0x10+0x02+0x01+0x02+0xC0+0x92
			Return：0x10+0x10+0x02+0xFC+0x04

Note: SLA = I2C address 0xB8. Table for the parity bit CRC, CRC for the 16-bit, low byte first, high byte in the post. Return error code: 0x80: does not support the function code 0x81: reading an illegal address 0x82: write data beyond the scope of 0x83: CRC checksum error 0x84: disable writes.

8、the license agreement

Without the copyright holder's prior written permission in any form or by any means, electronic or mechanical (including photocopying), any part of this manual may be reproduced, nor may its contents be communicated to third parties. The contents are subject to change without notice.

The company has a software and third-party ownership, the user only signed a contract or license the software before use.

9、warnings, and personal injury

Do not use this product as safety or emergency stop devices, as well as the product failure could cause personal injury to any other application, unless there are special purpose or authorized use. In the installation, handling, use or maintenance of the product, please consult the product data sheets and application notes. Failure to comply with this proposal, likely to cause death and serious injury. The Company will not assume the resulting personal injury or death of any compensation, and thus exempt from the company's managers and employees and affiliated agents, distributors, etc. that may arise from any claims, including: a variety of costs, compensation costs, legal fees and so on.

10、Quality Assurance

The company's direct purchase of their products to provide a period of 3 months quality assurance (from date of shipment). The company published the product data sheet specifications shall prevail. If the warranty period, the quality of the product proved defective indeed, the company will provide free repair or replacement. User must meet the following conditions:

① This product is found defective within 14 days of written notice to the Company;
② The product shall be paid by mail back to the company;
③ The product should be in the warranty period.

The Company applied only to those who meet the technical requirements of the occasion of the product resulting from defective products responsible. Those companies for their products are used in special applications without any guarantee, warranty or written statement. The company or its products into the product reliability of the circuit does not make any promises.