Trabajo Fin de Grado
Ingeniería de las Tecnologías de Telecomunicación

Implementación de una plataforma de automatización de bajo coste

Autor: José Manuel Martínez Rodríguez
Tutor: Federico Cuesta Rojo

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Implementación de una plataforma de automatización de bajo coste

Autor:
José Manuel Martínez Rodríguez

Tutor:
Federico Cuesta Rojo
Profesor Titular de Universidad

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Trabajo Fin de Grado: Implementación de una plataforma de automatización de bajo coste

Autor: José Manuel Martínez Rodríguez

Tutor: Federico Cuesta Rojo

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

A mi familia

A mis maestros

A mis amigos

AGRADECIMIENTOS

En primer lugar, agradecer a mis padres y a mi hermana por darme la oportunidad de estudiar esta carrera y ofrecerme todo su apoyo, paciencia y sobre todo sus ánimos para que pudiera conseguir mis objetivos y llegar hasta aquí. Gracias por creer en mí en todo momento.

A Mari Abreu, por todo el apoyo y ánimo que me ha dado cada vez que me caía por el camino, por ayudarme siempre que lo he necesitado y por armarse de gran paciencia conmigo.

A esos dos grandes amigos, Carlos y Jorge, que me motivaron a iniciar esta carrera y que gracias a ellos todo este recorrido ha sido mucho más fácil.

Por último, a Federico Cuesta por brindarme la oportunidad para realizar este trabajo fin de grado, aprender de él, y por la ayuda y dedicación durante el desarrollo del mismo.

Gracias a todos.

José Manuel Martínez Rodríguez

Sevilla, 2017

RESUMEN

La tendencia actual en la industria es la utilización de plataformas de automatización modulares. Estas plataformas están compuestas por dispositivos que se comunican entre sí a través de una red local.

El presente proyecto se basa en el diseño de una plataforma de automatización, la cual está constituida por un hardware de bajo coste, y será quien realizará la función de PLC (*Programmable Logic Controller*), y por un software para programar dicho PLC, que respete en todo momento el estándar internacional para la programación de PLCs, IEC 61131-3. De esta forma, conocido dicho estándar, cualquiera puede programar este PLC, cuyo coste será muy reducido comparado con los productos ofrecidos por las empresas líderes del sector. Además, esta plataforma es modular y permite la integración de múltiples dispositivos que pueden comunicarse entre sí de forma inalámbrica.

El elemento principal elegido para esta plataforma de automatización de bajo coste es una Raspberry Pi, más conocido por sus siglas en inglés SBC (*Single Board Computer*). Y el software empleado para programar dicha función de PLC sobre Raspberry Pi es Codesys. Todas las comunicaciones entre dispositivos se realizarán a través del protocolo de comunicaciones Modbus TCP/IP.

ABSTRACT

The current trend in the industry is the use of modular automation platforms. These platforms are integrated by devices that communicate with each other through a local network.

The present project is based on the design of an automation platform, which is constituted by a low cost hardware that will perform the PLC function (*Programmable Logic Controller*), and by a software to program that PLC, that respects in all the time the international standard for the programming of PLCs, IEC 61131-3. In this way, knowing this standard, anyone can program this PLC, whose cost will be greatly reduced compared to the products offered by the leading companies of the sector. In addition, this platform is modular and allows the integration of multiple devices that can communicate with each other wirelessly.

The main element chosen for this low-cost automation platform is a Raspberry Pi, called SBC (*Single Board Computer*). And the software used to program such a PLC function on Raspberry Pi is Codesys. All communications between devices will be made through the Modbus TCP / IP communications protocol.

Índice

Índice de Tablas

Índice de Figuras

1 INTRODUCCIÓN

El verdadero progreso es el que pone la tecnología al alcance de todos.

- Henry Ford -

El control y monitorización de plantas industriales y/o edificios a tiempo real aprovechando las herramientas de comunicación de las que disponemos puede suponer una serie de ventajas considerables tanto para usuarios, como para propietarios en aspectos de seguridad, confort, modularidad, ahorro energético y reducción de costes de mantenimiento.

En la actualidad, es primordial en la industria, y cada vez más en los edificios debido a la incorporación de sistemas domóticos, el poder revisar la productividad y reducir los errores en cualquier tipo de sistema gestionado por autómatas programables.

Existen diferentes protocolos para establecer las comunicaciones, los cuales permiten detectar anomalías y contribuir en la resolución del problema que se presente, e incluso hacer uso de sistemas SCADA, permitiendo así controlarlas de una manera más gráfica.

En el presente trabajo se tratará, desde un punto de vista teórico-práctico, el proceso de diseño e implementación de una planta de automatización de bajo coste mediante el uso de tecnologías de hardware libre y soluciones basadas en entorno web y comunicaciones de redes de datos.

1.1. Contexto

El proyecto está englobado dentro de un concepto llamado *Industry 4.0*, también conocido como la cuarta revolución industrial. Las llamadas fábricas inteligentes forman parte de *Industry 4.0*. Estas fábricas inteligentes son modulares y en ellas las máquinas, dispositivos, sensores y personas están conectados entre sí y se comunican unos con otros.

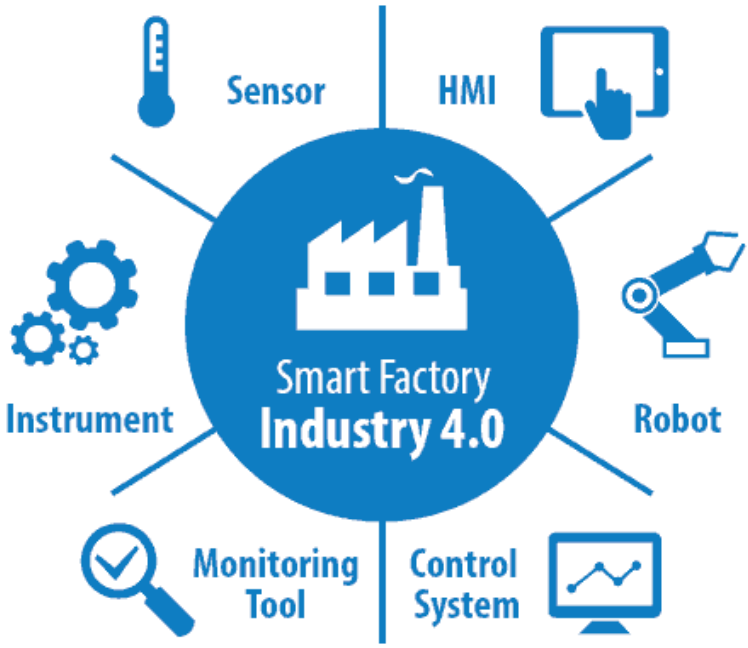

Figura 1 – Industry 4.0

Uno de los conceptos estrechamente relacionado con este trabajo, del que también depende *Industry 4.0*, es *Cyber-Physical Systems* (CPS). Este concepto está basado en la integración de la computación, las redes de comunicación y los procesos físicos. Ordenadores empotrados y redes de comunicación controlan y monitorizan el proceso físico, con bucles de realimentación en los que los procesos físicos afectan a la computación y viceversa. Dentro de la estructura modular de las fábricas inteligentes pertenecientes a *Industry 4.0*, los sistemas CPS monitorizan los sistemas físicos, crean una copia virtual y toman decisiones descentralizadas.

También podemos situar el proyecto dentro del concepto *Internet of things* (IoT). Este concepto está referido a la interconexión digital de objetos cotidianos con Internet, y supone un elemento clave dentro de Industry 4.0, ya que en las fábricas inteligentes las máquinas, dispositivos y sensores se comunican entre sí a través de IoT. Uno de los dispositivos más usados en proyectos relacionados con el IoT es Raspberry Pi. Su bajo coste, unido a la facilidad con la que se puede programar por estar basado en un software libre, lo convierten en un elemento clave del IoT.

La plataforma de automatización objeto de diseño en este trabajo, basada en Raspberry Pi, está conectada a Internet y por lo tanto podremos visualizar datos o enviar señales al proceso de forma remota. Además, una de las ventajas de un proyecto IoT es la modularidad, es decir, la capacidad de integrar más dispositivos que se comunicarán entre sí de forma inalámbrica. Ésta ventaja se verá reflejada en las diferentes pruebas y escenarios que comentaremos más adelante. Los conceptos IoT y CPS son similares, ya que comparten la misma estructura. Sin embargo, CPS presenta una mayor combinación y coordinación entre elementos físicos y computacionales. Por lo tanto, estamos hablando de sistemas complementarios como se puede ver en la siguiente figura:

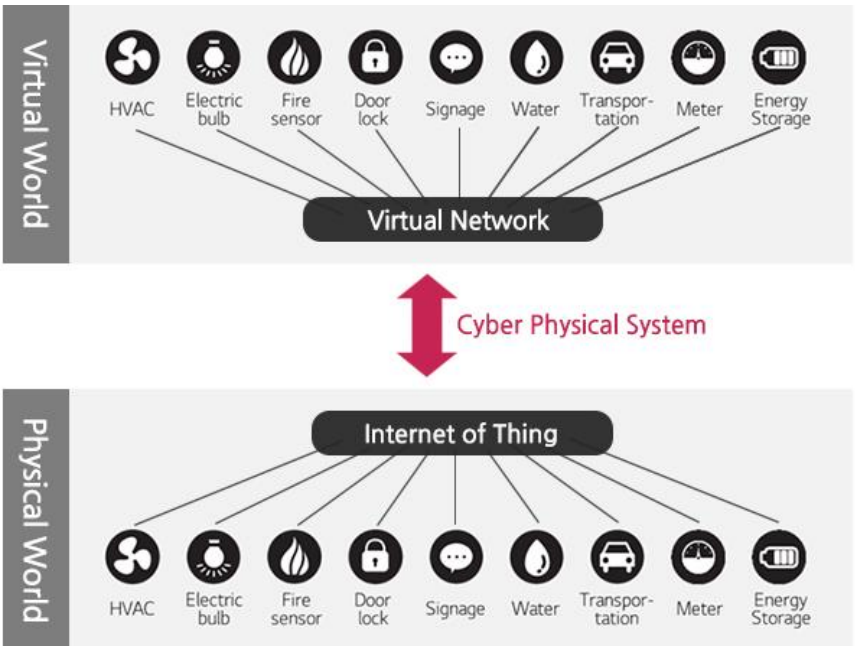

Figura 2 – Relación entre IoT y CPS

Por último, a nivel industrial cabe hablar de *Industrial Internet of Things* (IIoT) y *Cyber Physical Production Systems* (CPPS). El concepto IIoT, no es más que el uso de tecnologías IoT en la producción industrial como se observa en la siguiente figura. Este concepto está cada vez más presente en el mundo de la automatización industrial: una serie de dispositivos y sensores que se comunican entre sí y a su vez recogen y envían información a un servidor web que permite a los empleados detectar ineficiencias y problemas con rapidez, así como recopilar un registro histórico de datos de proceso.

Por otro lado, los sistemas CPPS crean una red inteligente de máquinas, tecnologías de comunicación e información, productos inteligentes y personas a lo largo de la cadena de valor y del ciclo de vida del producto.

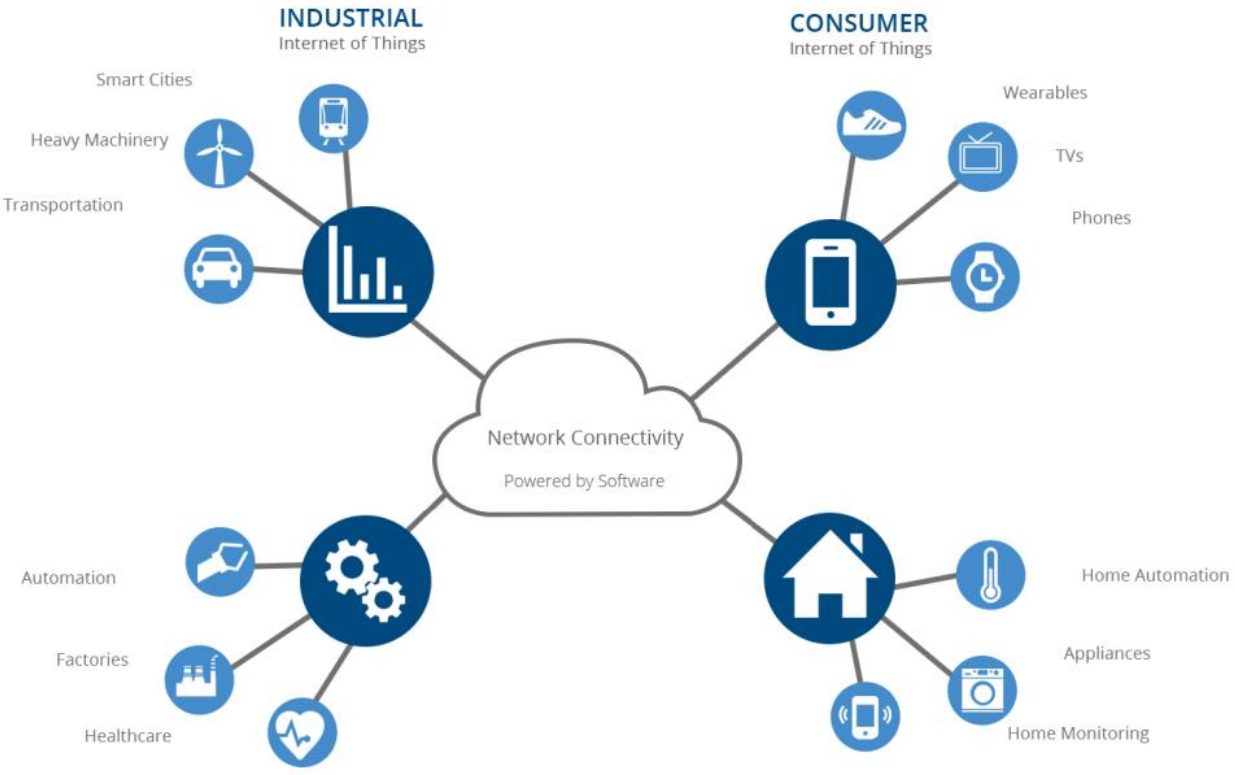

Figura 3 – Relación IIoT y IoT

1.2. Objetivos y alcance

El objetivo de este trabajo se basa en el diseño de una plataforma de automatización, constituida por un hardware de bajo coste que realice la función de PLC (*Programmable Logic Controller*), y por un software para programar dicho PLC de manera más gráfica y simple, respetando en todo momento el estándar internacional para la programación de PLCs IEC 61131-3. De esta manera, conocido dicho estándar, cualquiera puede programar este PLC de coste reducido. Además, esta plataforma es modular y permite la integración de múltiples dispositivos que puedan comunicarse entre sí a través de una red Ethernet o WiFi.

1.3. Organización del documento

Con objetivo de facilitar la lectura de este trabajo, se ofrece un breve resumen descriptivo de las partes que componen su documentación:

- **Introducción:** capítulo introductorio en el que se comenta brevemente el contexto en el que se encuentra este trabajo y los objetivos principales que se han planteado. También se indica la estructura seguida para la elaboración del mismo.
- **Protocolo de comunicaciones:** capítulo en el que se presenta el protocolo usado para realizar este trabajo, describiendo su funcionamiento, tipos de objetos, formato de la trama, e incluso las versiones del protocolo, entrando en más detalles en la versión usada, Modbus TCP/IP.
- **Hardware empleado:** capítulo en el que se describen los diferentes dispositivos usados a lo largo de todo este trabajo, detallando las características y especificaciones más relevantes.
- **Software empleado, CODESYS:** capítulo en el que se presenta el software empleado para realizar la implementación y tener el control del sistema de una manera más gráfica y sencilla.
- **Comprobación de funcionalidades básicas:** capítulo en el que se describen las distintas funcionalidades que se han ido implementando y mejorando, buscando la que más se adapta al objetivo del trabajo.
- **Escenario avanzado:** capítulo en el que se implementa un escenario más avanzado, añadiendo más funcionalidades en tiempo real y potenciando la interactuabilidad entre los dispositivos. También se describe la operabilidad desde una pantalla de explotación diseñada a través del mismo software.
- **Conclusiones y líneas futuras de trabajo:** resumen de las conclusiones obtenidas tras el desarrollo del trabajo. Además, se incluyen algunas ideas sobre futuras ampliaciones o mejoras que podrían desarrollarse a partir de este trabajo.
- **Referencias:** enumeración de las referencias utilizadas a lo largo del documento.
- **Glosario:** listado de siglas y abreviaturas a las que se hace mención en este documento.
- **Anexos:** se incluyen los anexos en los que se describe la preparación del entorno que sido necesaria para la realización de este trabajo. También se recoge información adicional de algunos puntos de este documento.

2 PROTOCOLO DE COMUNICACIONES MODBUS

Modbus es un protocolo de comunicaciones serie situado en el nivel 7 del Modelo OSI (*Open System Interconnection*), el cual fue desarrollado y publicado por Modicon (ahora conocido como Schneider Electric) para su gama de controladores lógicos programables, también llamados PLCs, en 1979.

Analizando el mercado industrial nos damos cuenta de que, actualmente, el protocolo Modbus es el protocolo de comunicaciones más instaurado en los dispositivos electrónicos industriales, sistemas de telecontrol y monitorización, lo que implica que, tanto a nivel local como a nivel de red, en su versión TCP/IP, seguirá siendo uno de los protocolos de referencias en las llamadas Smart Grids (*Redes inteligentes*), telecontrol, redes de sensores, y un extenso etcétera de sistemas de información que ya empiezan a asomar la cabeza en nuestro día a día.

El objetivo del protocolo Modbus consiste básicamente en la transmisión de información entre distintos equipos electrónicos conectados a su mismo bus. Existiendo en dicho bus un solo dispositivo maestro (Master) y varios equipos esclavos (Slaves) conectados [1].

En sus inicios estaba orientado a una conectividad a través de líneas serie como pueden ser RS-232 o RS-485, pero a lo largo del tiempo han aparecido variantes como es el Modbus TCP, el cual permite el encapsulamiento del Modbus serie en tramas Ethernet TCP/IP de forma sencilla. Esto sucede porque desde un punto de vista de la torre OSI, el protocolo Modbus se ubica en la capa de aplicación.

Las razones por las cuales el uso de Modbus destaca con respecto a otros protocolos de comunicaciones, y así haya conseguido ser el más estandarizado en el sector industrial, se debe a:

- Compatibilidad con dispositivos e instalaciones de viaja construcción: Su funcionalidad es tal que es capaz de funcionar sobre prácticamente la mayoría de medios de comunicación que incluyan cables de par trenzado, fibra óptica, Ethernet, o incluso conectividad inalámbrica como teléfonos celulares y microondas.
- Su implementación es simple: Requiere poco desarrollo debido a que, a diferencia de otros protocolos, se simplifica la estructura de las tramas y en consecuencia el acceso a los datos que no están almacenados en estructuras complejas.
- Es un estándar público y seguro: Esto resulta interesante para los fabricantes, ya que permite que estos puedan desarrollar dispositivos tanto Maestro como Esclavo sin gastos aplicados al protocolo. Este hecho facilita el acceso a la información y estructura del protocolo que, además, es muy básica pero funcional para su objetivo.
- Flexibilidad en el intercambio de información: Maneja bloques de datos sin suponer restricciones, es decir, la transmisión de información no está ligada a algún tipo de datos en concreto. Un ejemplo claro para entender el concepto: si se transmite un dato de 16 bits de información su representación no está sujeta a ninguna restricción, por lo que puede tratarse de un dato tipo Word con signo, un entero sin signo de 16 bits o la parte alta de una representación tipo Float de 32bits, etc. La representación del valor vendrá definida por la especificación que el fabricante dé del dispositivo, lo que permite la representación de un amplio rango de valores.

2.1. Funcionamiento

El protocolo Modbus siempre funciona con un Maestro y uno o más Esclavos, siendo el maestro quién controla en todo momento el inicio de la comunicación con los esclavos, solicitando información del resto de dispositivos conectados que ejercen como esclavos y son quienes suministran la información al primero. El esclavo por otro lado se limita a devolver los datos solicitados por el maestro.

Según la especificación, en una misma red puede haber hasta 247 dispositivos esclavos, esto es debido a que en una trama Modbus la dirección del esclavo se representa con un solo Byte, existiendo algunas direcciones reservadas para propósitos específicos como Broadcast [2].

0	From 1 to 247	From 248 to 255
Broadcast address	**Slave individual addresses**	**Reserved**

El nodo maestro no tiene ninguna dirección asignada, únicamente los nodos esclavos deben tener una dirección única en un bus y, a su vez, deben reconocer la dirección 0, que es la reservada como la dirección de difusión.

Las peticiones del nodo maestro a los nodos esclavos pueden ser:

- Modo Unicast: el maestro se dirige a un esclavo concreto e individual y éste, tras recibir y procesar la petición, devuelve una respuesta al maestro.

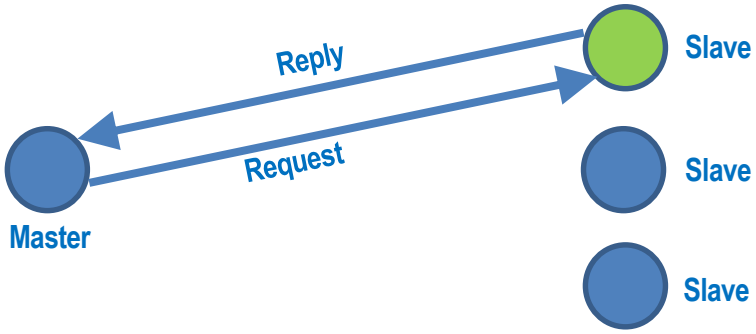

Figura 4 – Modo Unicast

- Modo Broadcast: el maestro se dirige a todos los esclavos que se encuentren en el mismo bus. La única restricción que tiene es que las peticiones de difusión solo pueden ser de escritura.

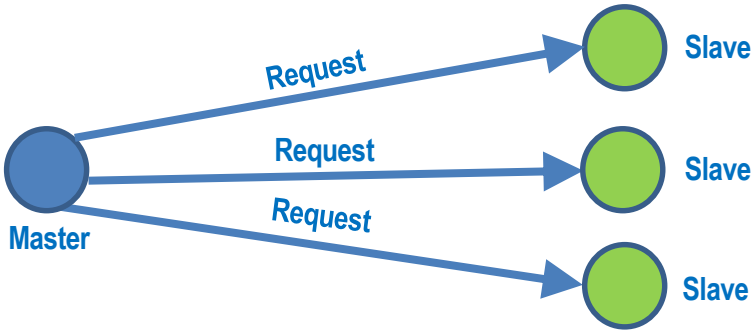

Figura 5 – Modo Broadcast

En definitiva, su funcionamiento se resume básicamente en: el Maestro pregunta y los Esclavos responden o

actúan en función de lo que este ordene.

2.2. Tipos de objetos

En el protocolo Modbus se distingue entre entradas digitales (discrete input), salidas digitales (coils), registros de entrada (input register) y registros de retención (holding registers). Tanto las entradas como las salidas digitales tienen una longitud de un bit, mientras que los registros, tanto de entrada como de retención, tienen una longitud de dos Bytes.

En la siguiente tabla se recoge tanto el acceso como el tamaño según el tipo de objeto proporcionado por un esclavo:

Object type	Access	Size
Discrete input	Read-only	1 bit
Coil	Read-Write	1 bit
Input register	Read-only	16 bits
Holding register	Read-Write	16 bits

Tabla 1 – Tipos de objetos. [3]

2.3. Formato de la trama

Los modos de transmisión definen como se envían los paquetes de datos entre maestros y esclavos, y el protocolo Modbus define dos variantes, con diferentes representaciones de los datos y de algunos detalles del protocolo ligeramente desiguales:

- Modbus RTU: La comunicación se realiza por medio de una representación binaria compacta de los datos. El formato RTU finaliza la trama con una suma de control de redundancia cíclica (CRC).
- Modbus ASCII: Es una representación legible del protocolo, pero menos eficiente. El formato ASCII utiliza una suma de control de redundancia longitudinal (LRC).
- Modbus TCP/IP: En realidad no es una variante más ya que es muy semejante al formato RTU, pero estableciendo la transmisión mediante paquetes TCP/IP a través del puerto 502.

Ambas implementaciones del protocolo, RTU y ASCII, son en serie.

La estructura de trama Modbus es muy simple, siendo uno de los motivos de su éxito junto a ser un protocolo abierto y a no estar orientado a conexión.

Por tanto, la estructura básica de una trama Modbus RTU, tanto de lectura como de escritura, es la que se muestra en la siguiente figura:

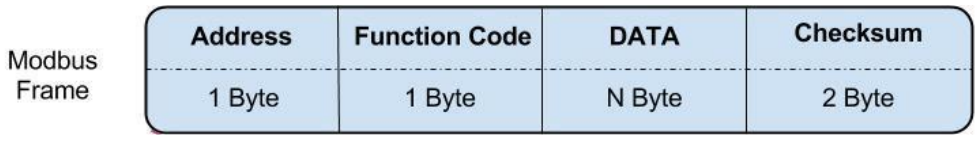

Figura 6 – Trama Modbus

- **Address**: Indica la dirección del esclavo que, al ser de 1 Byte, limita el número de esclavos que podemos tener conectados de forma correcta al bus serie Modbus como se indicó anteriormente en el punto 2.1.
- **Function Code**: En este campo se indica que acción requiere el maestro del esclavo al que va dirigida la trama. Este campo se particulariza dependiendo de si se trata de una trama maestro->esclavo o si por el contrario es esclavo->maestro.

  El código de operación puede tomar cualquier valor comprendido entre el 0 y el 127 (el bit de más peso se reserva para indicar error). Cada código se corresponde con una determinada operación. Algunos de estos códigos se consideran estándar y son aceptados e interpretados por igual por todos los dispositivos que dicen ser compatibles con MODBUS, mientras que otros códigos son implementaciones propias de cada fabricante. Es decir que algunos fabricantes realizan implementaciones propias de estos códigos "no estándar".

  La relación de funciones implementadas más usadas de este protocolo son las siguientes:

Function code	Function name
1	Read coils
2	Read Discrete Inputs
3	Read Multiple Holding Registers
4	Read Input Registers
5	Write Single Coil
6	Write Single Holding Registers
15	Write Multiple Coil
16	Write Multiple Holding Registers
23	Read/Write Multiple Registers

Tabla 2 – Códigos de función Modbus

- **Data**: Este campo dependerá tanto en contenido como en longitud de la función que se indique en el campo anterior, así como de si se trata de una trama maestro-esclavo o de respuesta esclavo-maestro.
- **Checksum**: Este campo consta de dos Bytes y sirve para la detección de errores en la trama. El CRC es un código más que frecuente en la detección de errores en redes digitales, sistemas de almacenamiento para la detección de modificación accidental de los datos o en este caso para comprobar la integridad de los datos en su transmisión por buses de campo.

2.4. Versiones del protocolo

En la actualidad, existen diversas versiones del protocolo Modbus, para el puerto serie, para el estándar de redes de área local como es Ethernet, e incluso otros protocolos que soportan el conjunto de protocolos TCP/IP de Internet. Esto es debido a que al igual que las redes de comunicaciones entre dispositivos electrónicos han ido evolucionando, han ido apareciendo variantes de este protocolo, en cuyos inicios estaba diseñado para redes sobre líneas serie. En la siguiente imagen recoge la perfecta armonía que existe entre las diferentes versiones, las cuales se explican seguidamente:

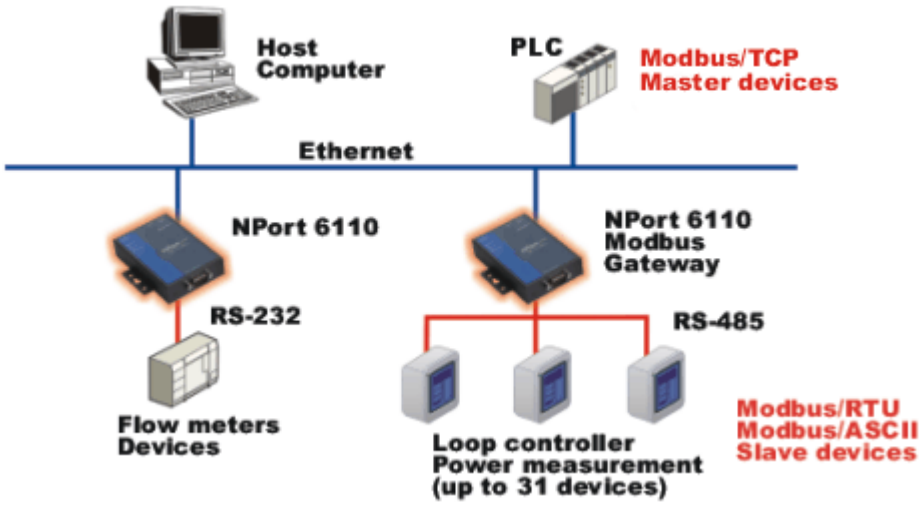

Figura 7 – Escenario con las diferentes versiones del protocolo

- **Modbus RTU**

  Como se ha comentado al inicio del punto 2, Protocolo de comunicaciones Modbus, es la implementación más instaurada para Modbus. Se utiliza en la comunicación serie convencional, usando una representación binaria compacta de los datos. Los mensajes Modbus RTU deben transmitirse continuamente sin oscilaciones entre caracteres. Permite la comunicación con 16 dispositivos esclavos por canal, a una velocidad de transferencia de hasta 19.2 Kbps.

- **Modbus ASCII**

  Esta versión utiliza igualmente la comunicación serie convencional, pero con la particularidad de que hace uso de caracteres ASCII. Al igual que la versión anterior, Modbus RTU, permite la comunicación con 16 dispositivos esclavos por canal, a la misma velocidad de transferencia, hasta 19.2 Kbps.

- **Modbus TCP/IP**

  Es la versión que se ha elegido para desarrollar este trabajo y por ello se va a entrar en más detalles sobre sus características y especificaciones [4].

  Esta variante es la evolución más fuerte, utilizada y conocida, ya que permite la implementación de este protocolo de comunicaciones sobre redes Ethernet, lo que supone un incremento en cuanto al grado de conectividad. Como se ha comentado anteriormente, es muy parecido al formato RTU, pero en este caso la transmisión se establece a través de paquetes TCP/IP mediante el puerto 502. Esta versión encapsula la trama base del protocolo Modbus en la capa 7 del modelo OSI, capa de aplicación, TCP/IP de forma sencilla.

  Por tanto, Modbus-TCP permite que se pueda usar en Internet, objetivo destacable por el que se inició su desarrollo, remitiéndose la especificación del protocolo a la IETF. De este modo, un dispositivo instalado en cualquier parte del mundo podría ser direccionado desde cualquier otra parte del mundo.

  También proporciona ventajas muy interesantes tanto para la empresa en cuestión como para el instalador:

  - Permite acceder remotamente utilizando un PC y así solucionar los problemas que se presenten, realizar reparaciones o incluso el mantenimiento de los mismos sin desplazamientos, lo que implica una mejora en el servicio con el cliente y una reducción palpable en los costes.

- Permite realizar la gestión de sistemas distribuidos geográficamente mediante el empleo de las tecnologías de Internet/Intranet actualmente disponibles.

Modbus TCP/IP se ha convertido en un estándar industrial debido a su simplicidad, reducido coste, requisitos hardware mínimos y por tratarse de un protocolo abierto, motivo por el cual hay un amplio catálogo de dispositivos Modbus TCP/IP en el mercado.

Gracias a la combinación de una red física versátil y escalable como el estándar universal de interredes TCP/IP y una representación de datos independientes de fabricante, proporciona una red y accesible para el intercambio de datos de proceso.

Esta versión del protocolo Modbus únicamente encapsula una trama Modbus en un segmente TCP, ya que este protocolo de transmisión proporciona un servicio orientado a conexión fiable, es decir, toda consulta espera una respuesta.

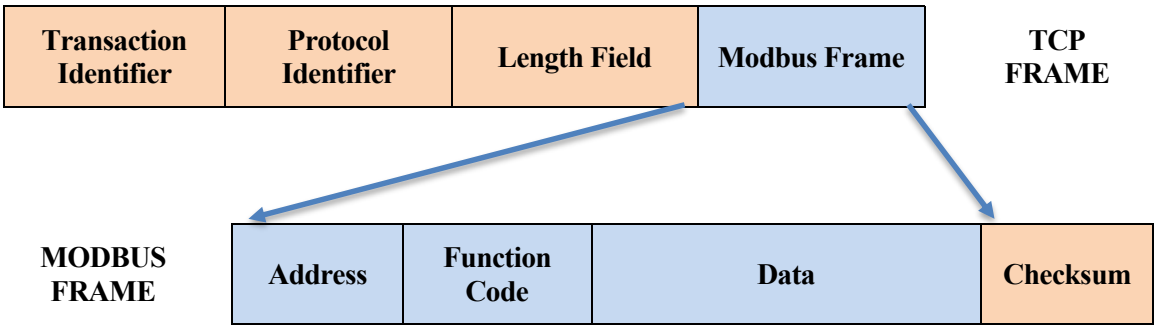

Figura 8 – Encapsulamiento de la trama Modbus en TCP

Esta técnica de consulta-respuesta encaja perfectamente con la naturaleza Maestro-Esclavo de Modbus, todo esto añadido a las ventajas que también proporcionan las redes Ethernet conmutadas a los usuarios en la industria. Modbus TCP/IP proporciona una solución para la gestión desde unos pocos a decenas de miles de nodos.

En cuanto a las prestaciones de un sistema Modbus TCP/IP depende básicamente de la red y el hardware usado. Si se usa sobre Internet, las prestaciones serán las correspondientes a tiempos de respuesta de Internet, por lo que pueden no ser las deseadas para un sistema de control, pero sí pueden ser suficientes para una comunicación destinada a la depuración y mantenimiento, reduciendo así el desplazamiento al lugar de la instalación. Si en cambio se usa sobre una Intranet de altas prestaciones con conmutadores Ethernet de alta velocidad, la situación es totalmente diferente.

Poniendo un ejemplo teórico, Modbus TCP/IP transporta datos con una eficiencia alrededor del 60% cuando se transfieren registros en bloque, y poniendo que 10BaseT proporciona unos 1,25 Mbps de datos, la velocidad de transferencia de información útil será:

$$1{,}25M / 2 * 60\% = 360000 \text{ registros por segundo}$$

Tomando 100BaseT la velocidad se incrementa en 10 veces. Todo este ejemplo es suponiendo que se están empleando dispositivos que pueden dar servicio en la red Ethernet aprovechando todo el ancho de banda disponible.

Además, debido al abaratamiento de los ordenadores personales y el desarrollo de redes Ethernet cada vez más rápidas, permite elevar las velocidades de funcionamiento, a diferencia de otros buses que están inherentemente limitados a una sola velocidad.

La comunicación entre dispositivos sobre Modbus TCP/IP es muy sencilla ya que solo requiere una pasarela que convierta el protocolo Modbus a Modbus TCP/IP.

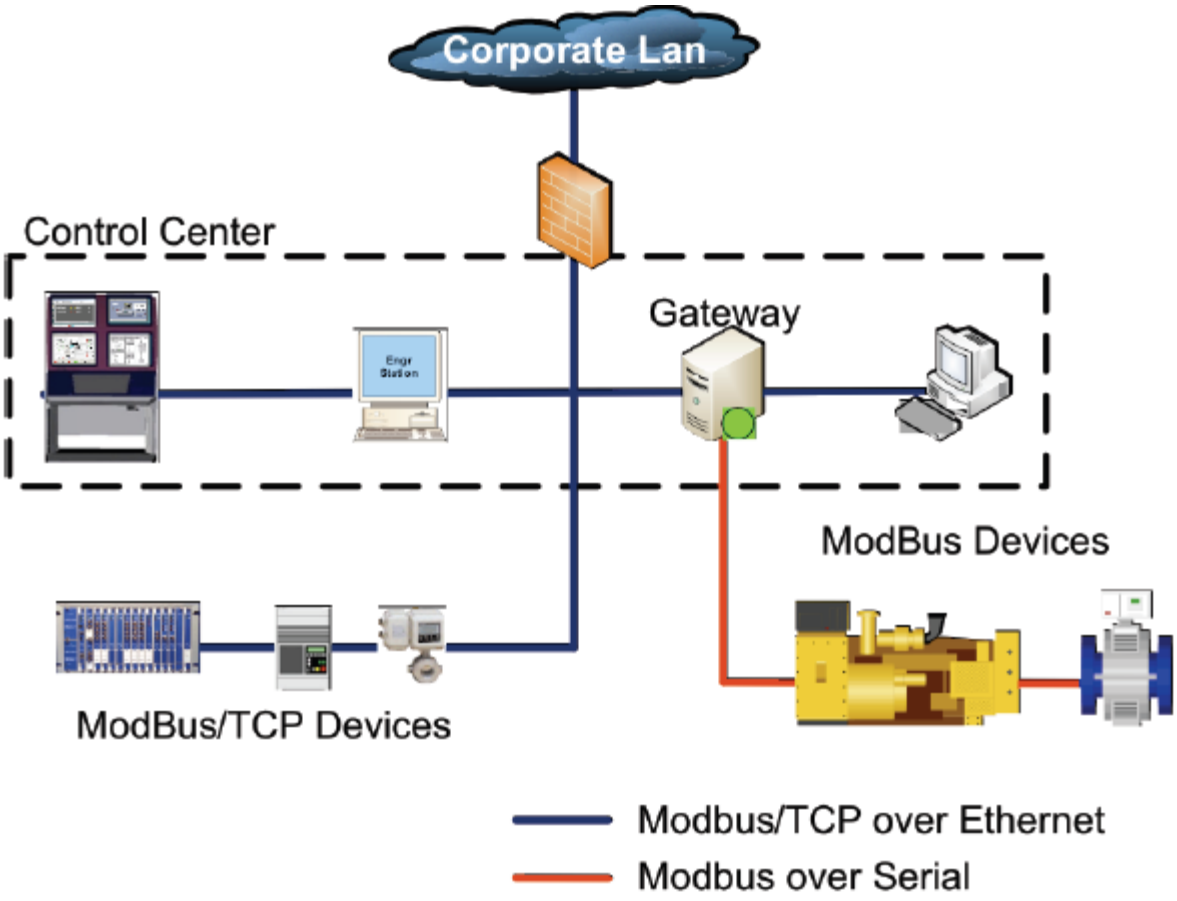

Figura 9 – Arquitectura Modbus TCP/IP

3 HARDWARE EMPLEADO

A continuación, se describen las principales características de los elementos hardware utilizados para la elaboración de este trabajo.

3.1. Raspberry Pi 3

Raspberry Pi [5] es una placa computadora de bajo coste desarrollada en Reino Unido por Raspberry Pi Foundation, una organización sin ánimo de lucro, con el objetivo de estimular la enseñanza de ciencias de la computación en los colegios y universidades.

Raspberry Pi surge con la idea de posibilitar "pequeños proyectos hardware" y el aprendizaje de la programación o de un lenguaje de programación, ya que los ordenadores que tenemos están orientados a tareas informáticas o de ocio, pero no cuentan con conexiones para cubrir este aspecto.

En cuanto al hardware, podemos encontrar varias versiones de Raspberry Pi:

- Raspberry Pi 1 Modelo A.
- Raspberry Pi 1 Modelo B y B+.
- Raspberry Pi 2 Modelo B.
- Raspberry Pi 3 Modelo B.

Esta última es la que más se comercializa en la actualidad, y cuenta con unas dimensiones de placa de 8.5cm por 5.3cm y presenta unas características muy interesantes como se pueden observar en la siguiente tabla [6]:

Raspberry Pi 3 Modelo B	
Chip integrado	Broadcom BCM2837 (CPU + GPU + DSP + SDRAM + Puerto USB)
CPU	1.2 GHz 64-bit Quad-core ARMv8
GPU	Broadcom VideoCore IV, OpenGL ES 2.0, MPEG-2 y VC-1 (con licencia), 1080p30 h.264/MPEG-4 AVC
Memoria SDRAM	1 GB (compartidos con la GPU)
Puertos USB 2.0	4
Entradas de vídeo	Conector MIPI CSI que permite instalar un módulo de cámara.
Almacenamiento integrado	MicroSD
Conectividad de red	10/100 Ethernet (RJ-45) vía hub USB, WiFi 802.11n, Bluetooth 4.1
Fuente de alimentación	5V vía microUSB o GPIO header
Periféricos de bajo nivel	17 GPIO y un bus HAT ID
Cosumo energético	800 mA (4.0 W)

Tabla 3 – Resumen de especificaciones Raspberry Pi 3 Modelo B

Además, esta placa de Raspberry Pi presenta una salida de audio y vídeo a través de un conector HDMI, lo que permite que podamos conectar la tarjeta tanto a monitores como a televisores que cuenten con una conexión de este tipo. También tiene una salida de video compuesto y una salida de audio a través de un conector minijack.

En la siguiente figura podemos ver cómo se encuentran distribuidos todos esos elementos:

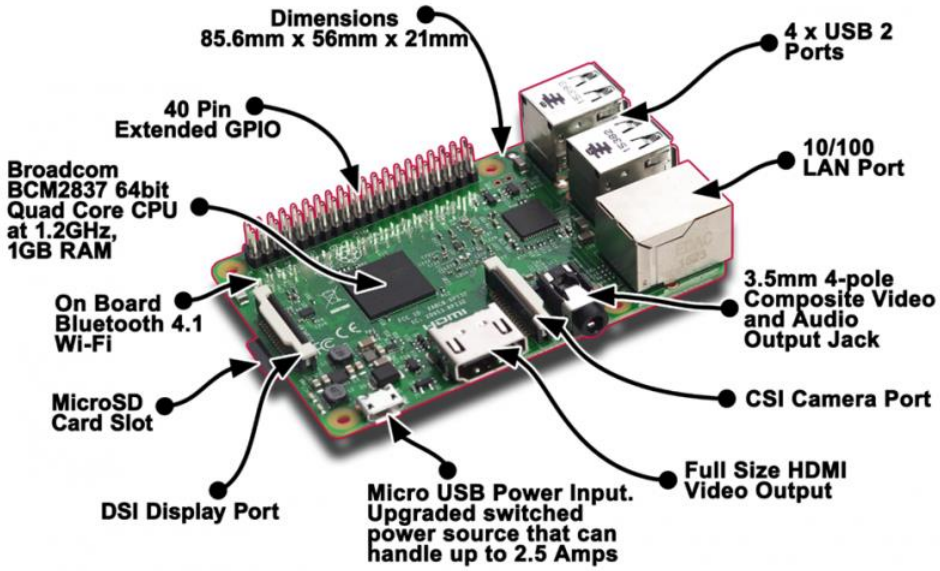

Figura 10 – Elementos de Raspberry Pi 3 Modelo B

Esta versión de Raspberry Pi ha sido elegida para el desarrollo de este proyecto por ser la más reciente y actual en el mercado, además de ofrecer mejores prestaciones con respecto a sus antecesoras a un precio muy similar. Al disponer de mejores especificaciones, se le puede asignar más carga de trabajo en paralelo sin que se vea afectado su rendimiento.

La funcionalidad que se le va a aplicar a asignar a esta placa es que permite correr un potente software para sistemas autómatas como es CodeSys [7] e incluso el uso de un sistema de visualización streaming en directo compuesto por una webcam y servidor web.

El sistema operativo elegido es Raspbian, que es una distribución del sistema operativo GNU/Linux y por tanto libre basado en Debian Jessie (Debian 8.0). Por comodidad a la hora de trabajar con la placa, se ha instalado el paquete SAMBA, el cual permite el intercambio de archivos entre Raspberry Pi 3 y Windows de una manera sencilla. También se le ha instalado y configurado el servicio No-IP para que la administración remota resulte mucho más fácil. Todo el proceso de instalación y configuración mencionado puede verse con más detalles en el anexo II de este documento, y en el anexo III se encuentra el pinout de esta placa.

3.2. Arduino

Arduino [8] es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar.

Por un lado, el hardware consiste en una placa con un microcontrolador y puertos de entrada/salida. Existen varios modelos de Arduino, cada uno de ellos con unas características específicas.

Por otro lado, el software consiste en la programación del microcontrolador en la placa Arduino mediante el lenguaje de programación Arduino, el cual está basado en Wiring [9], y el entorno de desarrollo, basado en Processing [10], sin embargo, también es posible utilizar otros lenguajes de programación populares como C, C++, C#, Java, Python, Ruby, etc., debido a que Arduino usa la transmisión serial de datos soportada por la mayoría de los lenguajes.

Arduino puede tomar información del entorno, a través de estos pines de entrada, de una amplia gama de sensores y puede afectar a aquello que le rodea, controlando motores, luces u otros actuadores.

Las placas pueden ser hechas a mano o compradas totalmente implementadas ya de fábrica y el software puede ser descargado de forma gratuita desde la web de Arduino [8]. Al tratarse de un hardware libre, todos sus diseños son abiertos y pueden ser reutilizados o incluso mejorados, convirtiendo a Arduino en una plataforma versátil para el desarrollo de productos electrónicos.

3.2.1. Arduino UNO

La placa Arduino UNO es de las más conocidas y utilizada para introducirse en el aprendizaje de este dispositivo. En particular, la que se ha empleado para realizar los primeros pasos en este proyecto es la versión R3.

Figura 11 – Vista frontal y trasera Arduino UNO Rev3

Sus especificaciones principales son [11]:

Arduino UNO	
Microcontrolador	ATmega328
Voltaje de operación	5V
Voltaje de entrada recomendado	7-12V
Pines digitales E/S	14 (6 disponen de salida PWM)
Pines de entrada analógica	6
Consumo por pin E/S	40 mA
Consumo del pin 3.3V	50 mA
Memoria Flash	32kB (0.5kB empleados por el bootloader)
EEPROM	1 kB
SRAM	2 kB
Frecuencia de reloj	16 MHz

Tabla 4 – Resumen especificaciones Arduino UNO Rev3

Cuenta con puerto de comunicaciones UART (serie hardware), USB (puerto virtual), comunicación mediante I2C y SPI. El pinout de esta placa puede encontrarse en el anexo III.

- ***Shield Ethernet***

El Arduino Shield Ethernet [12] nos da la capacidad de conectar un Arduino a una red Ethernet. Es la parte física que implementa la pila de protocolos TCP/IP.

Está basada en el chip ethernet Wiznet W5100, el cual provee de una pila de red IP capaz de soportar TCP y UDP. Soporta hasta cuatro conexiones de sockets simultáneas. A través de la librería Ethernet [13] le permite leer y escribir los flujos de datos que pasan por el puerto ethernet, por lo que permitirá escribir sketches que se conecten a internet usando la Shield.

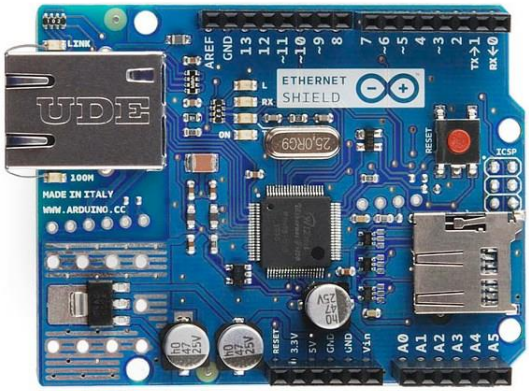

Figura 12 – Vista frontal Shield Ethernet

El shield provee un conector ethernet estándar RJ45. También dispone de unos conectores que permiten conectar a su vez otras placas encima y apilarlas sobre la placa Arduino.

La placa Arduino se comunica con el módulo W5100 y la microSD utilizando el bus SPI (mediante el conector ICSP). Esto se encuentra en los pintes digitales 11, 12 y 13 en el modelo UNO. El pin 10 es utilizado para seleccionar el W5100 y el pin 4 para la microSD. Estos pines no pueden ser utilizados para otros fines mientras la Ethernet Shiel esté conectada, hay que tenerlo en cuenta al escribir el código.

El botón de reset en el Shield resetea ambos, el W5100 y la placa Arduino.

Contiene varios LEDs para información:

- ON: indica que la placa y la Shield están alimentadas.
- LINK: indica la presencia de un enlace de red y parpadea cuando la Shield envía o recibe datos.
- 100M: indica la presencia de una conexión de red de 100 Mb/s (de forma opuesta a una de 10Mb/s).
- RX: parpadea cuando el Shield recibe datos.
- TX: parpadea cuando el Shield envía datos.

Dispone de un zócalo para tarjetas de memoria microSD en la que se puede almacenar ficheros o para utilizarlo como servidor web embebido. También incluye un controlador de reset automático para que el chip interno W5100 esté bien reiniciado y listo para utilizar al arranque.

3.2.2. Arduino Yún

El Arduino Yún es el primer miembro de una nueva serie de placas Arduino que combinan la potencia de Linux junto con la sencillez característica de Arduino. Combina el chip del modelo Leonardo (ATMega32U4) junto con un módulo SOC (System-On-a-Chip) corriendo una distribución de Linux llamada Linino, basada en OpenWRT. Una de las características más interesantes es que soporta red cableada Ethernet y Wifi.

Figura 13 – Vista frontal y trasera Arduino Yún

El chip Arduino está conectado al módulo Linux, por lo que es muy fácil que se comuniquen entre ambos y delegar procesos pesados a la máquina Linux integrada en la placa.

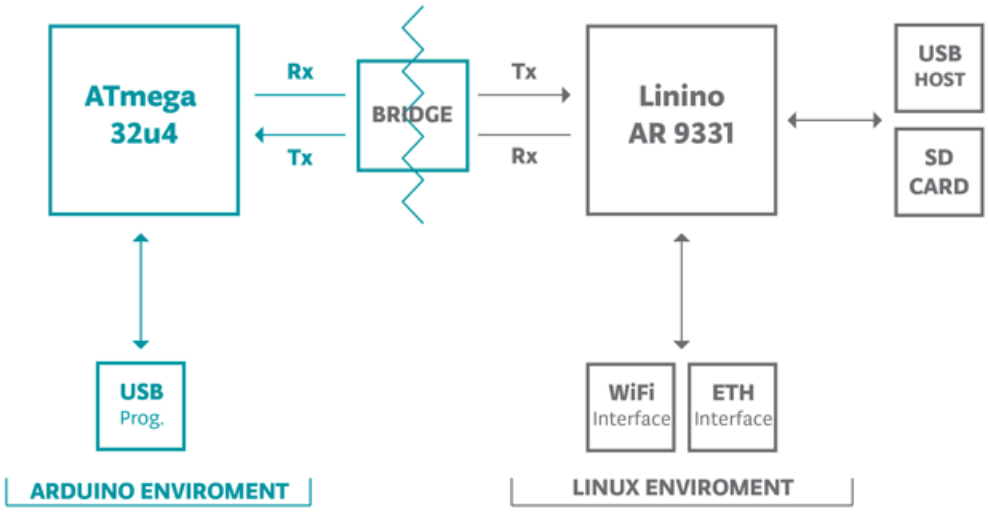

Figura 14 – Esquema por bloques de la comunicación ATmega32u4 y Linino AR 9331

En cuanto a la conectividad, dispone de dos conexiones de red. Una red ethernet 10/100 mbps y otra Wifi (IEEE 802.11 b/g/n, 2,4GHz) que puede montarse como cliente o como punto de acceso.

La comunicación entre procesadores, ATMega32U4 con el módulo Linux, se realiza a través de la librería Bridge [14], que facilita mucho las cosas y es soportada directamente por el team de Arduino. El puerto serial del AR9331 está conectado al serial del 32U4 con los pines 0 y 1. El puerto serie del AR9331 es un acceso a la consola CLI, lo que permite lanzar procesos y recuperar mensajes directamente desde la consola. Varios paquetes de gestión del sistema de archivos y administración ya están preinstalados por defecto, entre ellos el intérprete de Python. y la librería bridge permite también instalar y lanzar aplicaciones propias con ese mismo sistema.

Una de las ventajas más interesantes, es que la placa (del lado del 32U4) puede ser programada directamente por Wifi a través del módulo Linux.

También cabe destacar que dispone de un zócalo para memoria MicroSD que permite almacenar datos en ella como páginas web, datos logados o cualquier otra cosa que necesitemos, ampliando aún más las posibilidades de la placa.

Respecto a sus especificaciones principales, las dividimos en las del microcontrolador Arduino y el microprocesador Linux [15]:

Microcontrolador Arduino	
Microcontrolador	ATmega32u4
Voltaje Operativo	5V
Voltaje de Entrada	5V
Pines digitales de entrada/salida	20
Canales PWM	7
Canales de Entrada Analógica	12
Corriente DC en pines 5V	40 mA
Corriente DC en pines 3.3V	50 mA
Memoria Flash	32 KB (4 KB usados por bootloader)
SRAM	2.5 KB
EEPROM	1 KB
Frecuencia de reloj	16 MHz

Tabla 5 – Resumen especificaciones Microcontrolador Arduino Yún

Microprocesador Linux	
Procesador	Atheros AR9331
Arquitectura	MIPS @400MHz
Voltaje Operativo	3.3V
Ethernet	IEEE 802.3 10/100Mbit/s
WiFi	IEEE 802.11 b/g/n
USB Tipo-A	2.0 Host/Device
Lector MicroSD	Micro-SD
RAM	64 MB DDR2
Memoria Flash	16 MB

Tabla 6 – Resumen especificaciones
Microprocesador Arduino Yún

Como todos los modelos de Arduino, el Yún posee diferentes interfaces para entrada/salida de datos:

- Comunicación Serial: 0 (RX) and 1 (TX)
- TWI: 2 (SDA) and 3 (SCL)
- PWM (8 bits): 3, 5, 6, 9, 10, 11 y 13
- SPI: en los pines ICSP

El Arduino Yún cuenta también con algunos LEDs de estado que indican el estatus del dispositivo en un determinado instante:

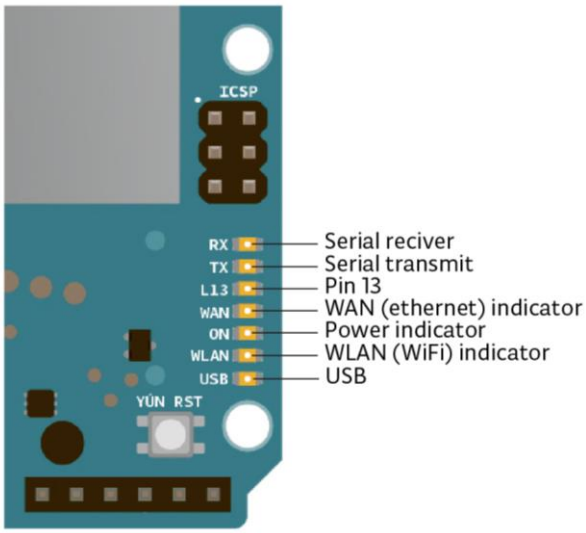

Figura 15 – Indicadores LEDs de estado Arduino Yún

También cabe destacar que el Arduino Yún cuenta con 3 botones de reset, uno para el microcontrolador, uno para el microprocesador y otro para el Wifi:

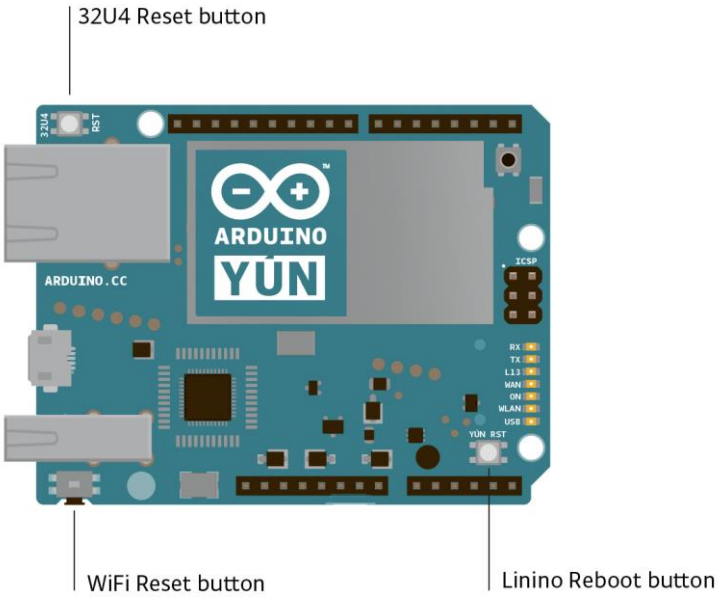

Figura 16 – Botones Reset Arduino Yún

El pinout de esta placa puede encontrarse en el anexo III.

3.3. Router

Un router es un dispositivo que proporciona conectividad a nivel de red o nivel 3 en el modelo OSI, cuya función principal consiste en enviar o encaminar paquetes de datos de una red a otra, es decir, interconectar subredes, siendo estas subredes un conjunto de máquinas IP que se pueden comunicar sin la intervención de un encaminador y que por tanto tienen prefijos de red distintos.

En este proyecto se ha decidido usar este dispositivo ya que se pretende tomar el control y mantenimiento del sistema a automatizar remotamente, es decir, desde el exterior a la red local. Además, también dispone de conexión inalámbrica, lo que permite que el sistema sea mucho más versátil y compatible con una amplia gama de dispositivos.

El router elegido es del productor Observa Telecom modelo Home Station ADSL BHS-RTA [16] por ser el dispositivo del que se dispone, pero puede ser cualquier otro router de cualquier marca y/o modelo con idéntico resultado.

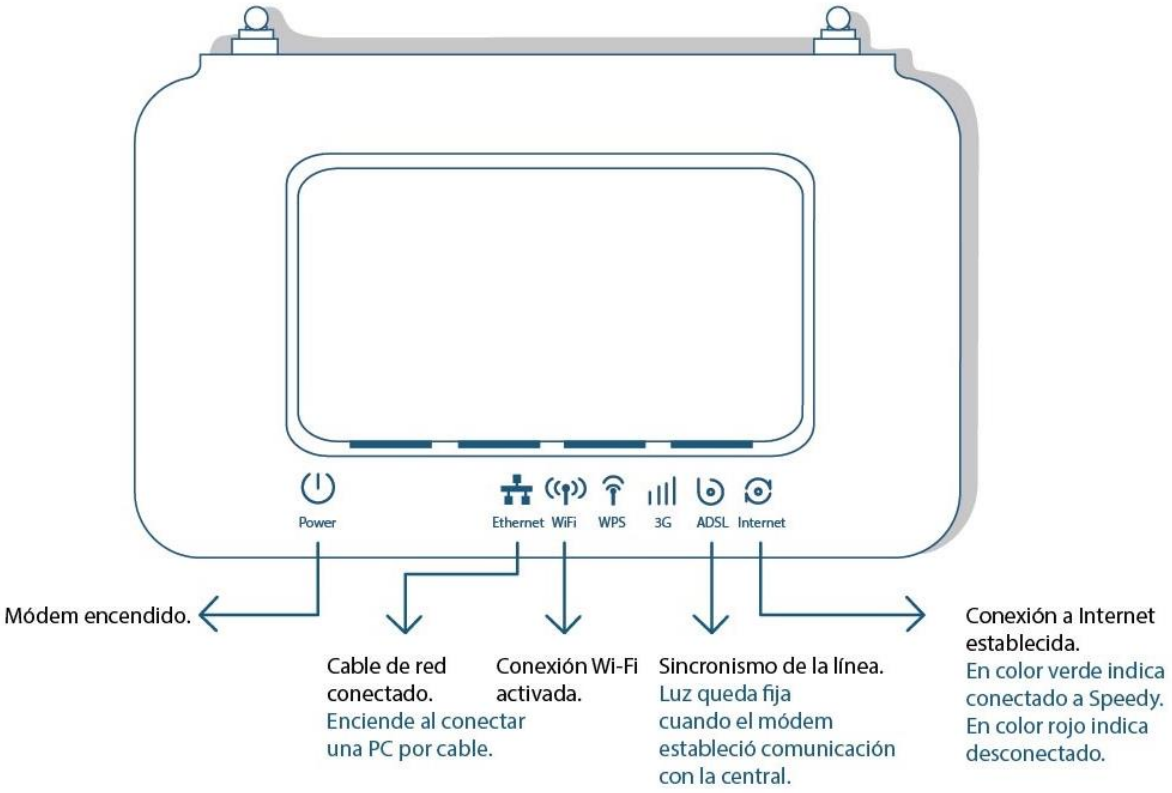

Figura 17 – Vista frontal Router Observa Home Station BHS-RTA

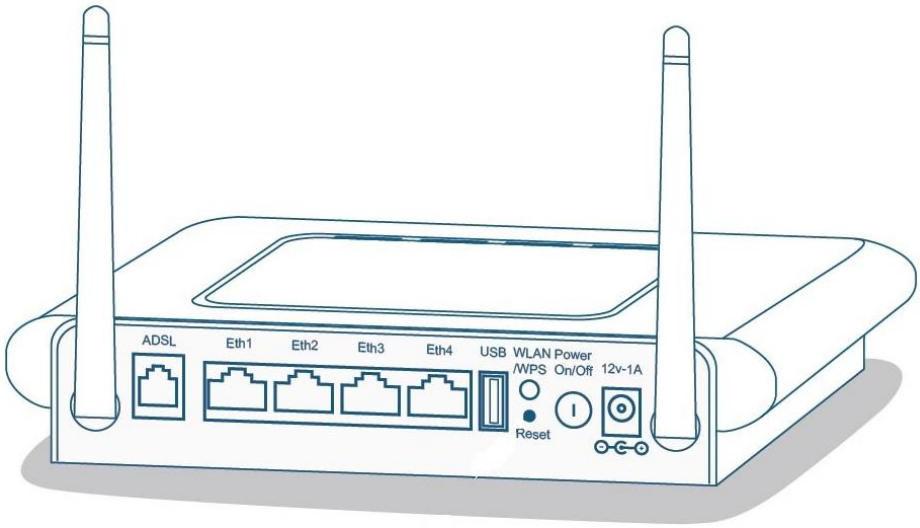

Figura 18 – Conexionado Router Observa HomeStation BHS-RTA

La configuración del router va a consistir básicamente en la configuración del WiFi para poder aprovechar la conectividad inalámbrica, reserva de direcciones IPs para los dispositivos del proyecto, evitando así conflictos de IPs y, por último, la apertura de puertos para poder tener un control y acceso remoto desde una red ajena a la red local.

3.4. Webcam

Una webcam es un dispositivo de entrada el cual permite, entre otras funciones, capturar imágenes y transmitirlas a través de Internet vía streaming. Ese es el fin y funcionalidad que le vamos a dar en este proyecto, capturar para visualizar el estado inmediato del sistema implementado.

La webcam usada para el proyecto es el modelo Logitech QuickCam E 2500 por ser el dispositivo del que se dispone, pero puede ser cualquier otra webcam, cámara de vigilancia, … de cualquier marca y/o modelo con idéntico resultado.

Figura 19 – Webcam usada en el proyecto

Su conexión es a través del puerto USB, por lo que es posible usarla en cualquier computador, embebido o no, a través de este conector. En nuestro caso, irá conectada la Raspberry Pi ya que dispone de este puerto.

Para realizar la captura y streaming del sistema en tiempo real, se ha instalado y configurado el paquete "*mjpg-streamer*" disponible desde el repositorio de Raspberry. Todo este proceso de instalación y configuración se encuentra con más detalles en el anexo II de este documento.

4 SOFTWARE EMPLEADO: CODESYS

Figura 20 – Software CoDeSys

CodeSys [7] es un software comercial desarrollado por la empresa alemana 3S-Smart Software Solutions GmbH y actualmente es la plataforma más extendida en Europa para el desarrollo de aplicaciones de acuerdo a la norma IEC-61131-3, cuyos lenguajes definidos en ella son:

- Lenguajes de texto:
  - IL (Lista de instrucciones): Lenguaje de programación parecido al lenguaje ensamblador.
  - ST (Texto estructurado): Similar a la programación en C.
- Lenguajes gráficos:
  - LD (Diagrama Ladder): Permite al programador combinar contactos de relé y las bobinas.
  - FBD (Diagrama de bloques de función): Permite programar rápidamente, tanto expresiones como en lógica booleana.
  - SFC (Bloques de función secuenciales): Interesante para los procesos de programación secuencial.
  - CFC (Gráfico continuo de funciones): Editor orientado a FBD donde las conexiones entre las entradas, salidas y los operadores se fijan automáticamente.

Este entorno de programación permite realizar todos los pasos necesarios para el diseño de aplicaciones para autómatas programables de diferentes fabricantes, entre ellos facilita la edición de programas en los lenguajes de que propone la citada norma, la depuración de programas, la simulación, la comunicación con el PLC y la monitorización del programa que se ejecute en dicho dispositivo.

Codesys organiza toda la información de cada aplicación que se va a ejecutar en el autómata programable en "Proyectos". Un proyecto incluye una o varias POUs, es decir, toda la información necesaria para la configuración del PLC que se esté usando, las variables globales, interfaces gráficas para controlar y visualizar la ejecución del programa, etc.

Respecto al código, un proyecto puede incluir una o varias POUs como antes se ha mencionado, cada una de ellas pueden estar escritos en lenguajes diferentes y puede ser de cualquier tipo: Programas, Funciones o Bloques Funcionales.

4.1. Descarga e instalación

Para la instalación de este software [17], se ha acudido a la web del desarrollador (http://www.codesys.com/), concretamente a la sección "Download". Para poder descargar este software es necesario registrarse previamente y tras ello comienza la descarga. Para la realización de este proyecto se ha utilizado la versión 3.5 SP10 de Codesys que es la versión más reciente encontrada.

La interfaz de Codesys V3 se muestra en la siguiente figura:

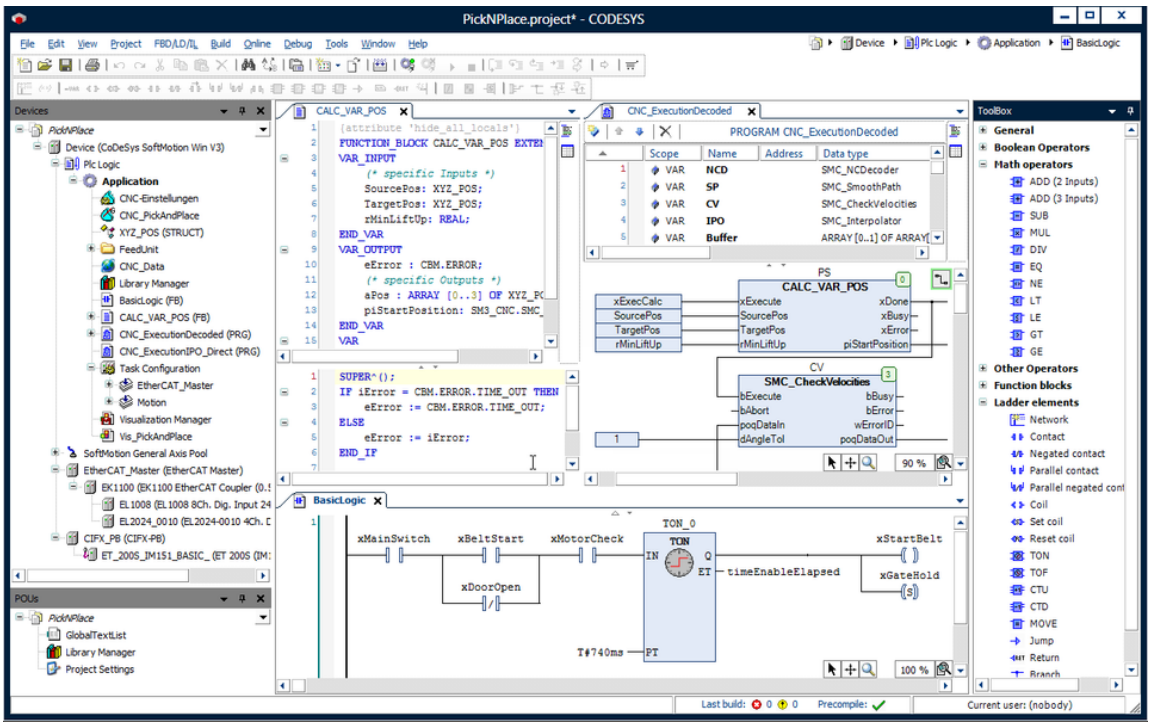

Figura 21 – Interfaz Codesys V3

4.1.1. Añadir runtime de Codesys en Raspberry Pi

Tras la instalación del software, hay que instalar un runtime de Codesys para Raspberry Pi. Este runtime es necesario ya que el objetivo es diseñar un programa en el entorno de Codesys para PC, cargar este programa en Raspberry Pi, y a partir de ahí prescindiremos del PC (salvo que queramos visualizar datos del proceso) ya que el programa quedará guardado en Raspberry Pi, que de ahora en adelante se comporta como un PLC.

El software Codesys se encarga de transferir a Raspberry Pi todos los ficheros necesarios para la instalación y configuración, siendo éste un proceso necesario únicamente para la preparación del entorno, es decir, una vez transferido e instalado, cada vez que se arranque Raspberry Pi, se ejecuta automáticamente el runtime de Codesys con el último programa cargado.

El paquete descargado para Raspberry Pi no se instala directamente sobre ésta, si no sobre el entorno de desarrollo de Codesys para PC. En dicho entorno, abrimos el administrador de paquetes (*Herramientas/Administrador de paquetes…*) e instalamos el paquete de Codesys para Raspberry Pi. Una vez instalado el paquete en el entorno de programación, es hora de instalar el runtime en la Raspberry Pi. Para ello, en primer lugar, encendemos la Raspberry Pi, que debe estar conectada a Internet.

4.2. Monitorización

Aunque una vez cargado el programa en Raspberry Pi ya no es necesario el IDE para PC, si resulta bastante útil como herramienta de monitorización. Existen varias formas de monitorizar los resultados:

- Si tras cargar el programa mantenemos abierto el entorno de desarrollo para Codesys, podemos visualizar en tiempo real que partes del programa se están ejecutando y que variables están activas.
- Otra opción es a partir del paquete "*CODESYS Visualization*", disponible en la web del fabricante, el cual trae una serie de herramientas gráficas para crear visualizaciones y mostrar resultados.

  Está opción está muy bien ya que se puede usar bien a través del IDE de Codesys o bien transferirlo a Raspberry Pi para que lo muestre a través de una interfaz web, todo ello sin necesidad de tener activo el entorno de desarrollo de Codesys. En este proyecto se hará uso de esto último, para conseguir así controlar remotamente el sistema desde cualquier lugar.

  Como se puede observar en las siguientes figuras, la visualización gráfica es la misma, solo que en la de la izquierda es necesario el entorno de desarrollo de Codesys y en la derecha no, se muestra la interfaz web a través del navegador.

Figura 22 – Monitorización y Pantalla de explotación en: a) Codesys y b) Interfaz web

5 COMPROBACIÓN DE FUNCIONALIDADES BÁSICAS

En este capítulo se va explicar algunas de las comunicaciones y pruebas que se han realizado hasta conseguir entender y configurar el protocolo Modbus en diferentes escenarios.

5.1. Comunicación entre Raspberry - Arduino por Puerto Serial (USB)

En esta primera prueba, se realiza una comunicación entre Raspberry y Arduino UNO aprovechando el conector que ambos comparten como es el USB y empleando Python [18]. Se ha elegido Arduino UNO como se podría haber elegido Arduino Yún obteniendo la misma funcionalidad y resultado.

En primer lugar, en necesario instalar el paquete Python Serial en Raspberry Pi para la comunicación serial con Arduino. Para ello, se accede al terminal de Raspberry Pi por SSH y se ejecuta el siguiente comando:

- `sudo apt-get install python-serial`

A continuación, se implementa el escenario, el cual consiste básicamente en conectar Raspberry Pi con Arduino a través del conector USB y conectar en uno de los pines de Arduino un led. La forma correcta de conectar el correctamente el led es colocando en serie con el terminal largo (ánodo) una resistencia de al menos 220 ohmios y ésta por ejemplo al pin 13, y el otro terminal corto (cátodo) del led a tierra, pin GND de Arduino UNO. En la siguiente figura se muestra cómo debe quedar el escenario:

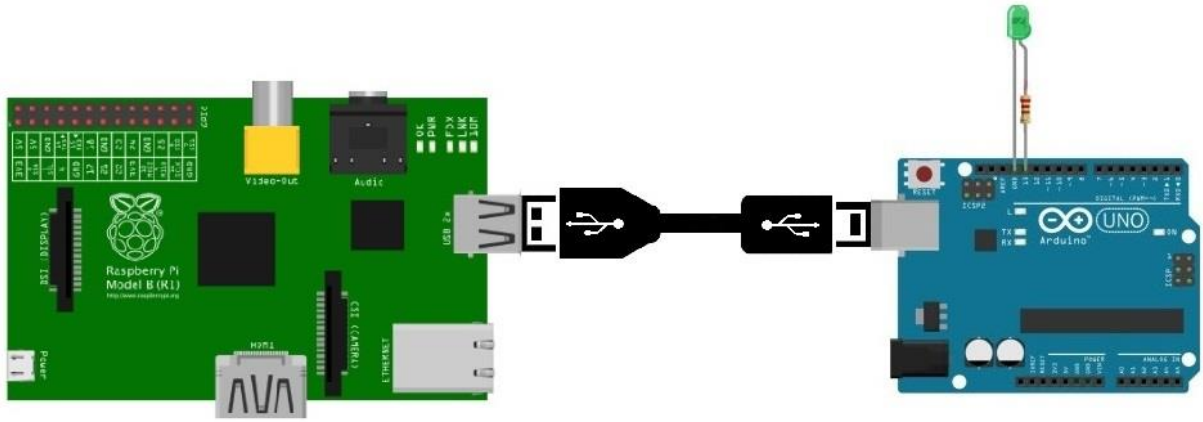

Figura 23 – Escenario de prueba de comunicación entre Raspberry Pi y Arduino UNO por puerto serial, USB

Tras tener el escenario montado, implementamos el siguiente código fuente para Arduino a través del propio Arduino IDE. Su funcionalidad es muy simple, si recibe por el puerto serial la letra H enciende el led, y viceversa si recibe la letra L.

```
/*Programa para hacer que Arduino espere ordenes de Raspberry Pi*/

int led = 13;                          //Declaramos el pin digital del LED

void setup () {
   pinMode(led, OUTPUT);               //LED 13 como salida
   Serial.begin(9600);                 //Inicializo el puerto serial a 9600
                                       baudios
}

void loop () {
   if (Serial.available()) {           //Si está disponible
      char c = Serial.read();          //Guardamos la lectura en un char
      if (c == 'H') {                  //Si es una 'H', enciendo el LED
         digitalWrite(led, HIGH);
      } else if (c == 'L') {           //Si es una 'L', apago el LED
         digitalWrite(led, LOW);
      }
   }
}
```

Ahora tan solo falta transmitir dichas letras por el puerto serial desde Raspberry Pi, y para ello ejecutamos a través del terminal el siguiente comando:

- `ser.write('n')`

El resultado es el esperado pero esta funcionalidad no resulta factible para un proyecto de gran envergadura, ya que se encuentra limitado por el número de puertos serial (USB).

5.2. Comunicación Maestro - Esclavo entre Arduinos a través de librerías

En esta segunda prueba, se realiza una comunicación entre Arduinos, siendo uno de ellos Maestro y el resto esclavos. Para ello, se hace uso del protocolo Modbus a través de las librerías correspondientes para Maesrto-Esclavo. "*MgsModbus.h"* y "*MgsModbus.cpp"*, que se pueden encontrar en el anexo I.

El escenario que se implementa para esta comprobación de funcionalidad es muy simple, únicamente consiste en realizar la conexión entre Arduinos y actuar sobre el esclavo mediante el monitor serie del maestro. Además, se añade un led en cada uno de los Arduinos los cuales se encienden cuando se transmite un '1' o se apagan cuando es un '0'. Todo ello se puede ver en la siguiente figura:

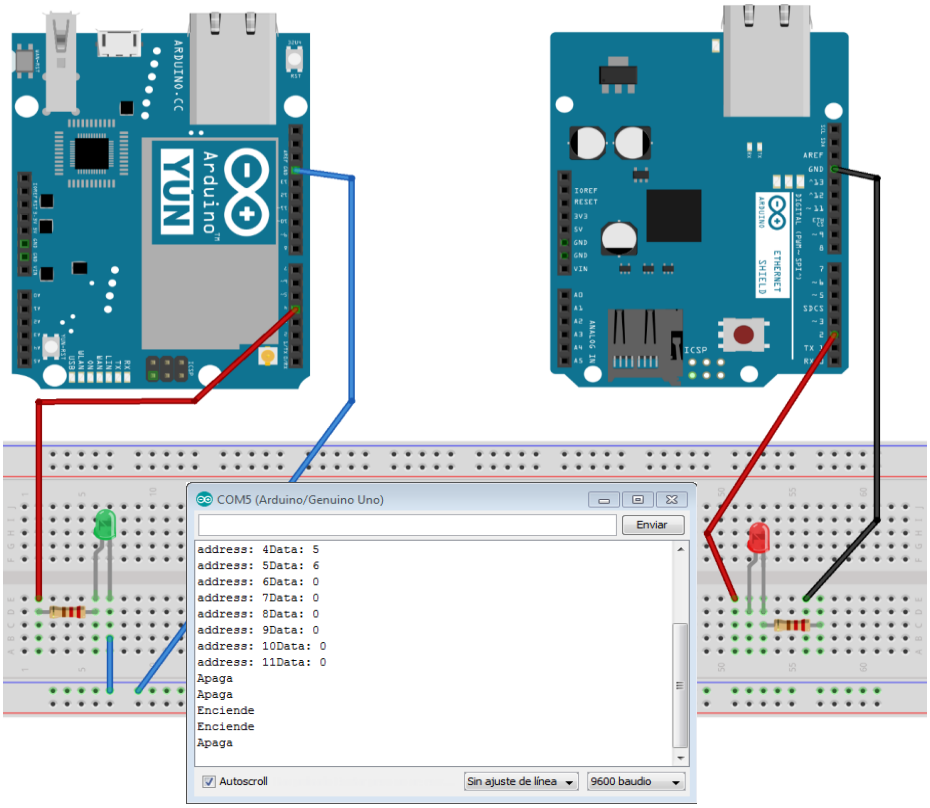

Figura 24 – Escenario de prueba de comunicación Modbus entre Arduinos

Tras tener el escenario montado, implementamos los siguientes códigos fuente en ambos Arduinos a través del propio Arduino IDE. Su funcionalidad es muy simple, si se transmite por el monitor serie del master un '1' se encienden y si es un '0' se apagan.

Este caso está muy bien si se trata de una aplicación con una configuración sencilla, ya que la configuración y programación se realiza a través de código fuente en Arduino y haciendo uso de librerías, pero para un proyecto mayor no resulta tarea simple, por tanto, se busca algo más gráfico y fácil de configurar y mantener en caso de problemas.

5.3. Comunicación entre Codesys - PLC en Windows - Arduinos

En esta tercera prueba, se hace uso de un entorno más gráfico y configurable como es el software descrito en el punto 4, Codesys. Básicamente permite implementar el escenario de la aplicación elegida de una manera mucho más intuitiva y, en este caso, además hace que Windows actúe como un controlador lógico programable, PLC.

Como en el punto anterior, 5.2, hacemos uso del protocolo Modbus para las comunicaciones. En este caso, el maestro sería la propia máquina local en la que se usa el software Codesys, ya que tiene la gran utilidad de desplegar un PLC de manera virtual sobre Windows. Los esclavos serían tres Arduinos, uno de ellos Arduino UNO y los otros dos Arduinos Yún, comunicándose estos dos últimos de forma distinta con el maestro, uno sería por Ethernet y el otro a través de WiFi.

El escenario para esta prueba de funcionalidad es muy simple, cada Arduino tiene conectado un led que será sobre el que actúa el maestro. La forma correcta de conectar un led en Arduino es colocando en serie con el terminal largo (ánodo) una resistencia de al menos 220 ohmios y ésta a un pin digital de Arduino, y el otro terminal corto (cátodo) del led a tierra, pin GND. En la siguiente figura se muestra como debe quedar el escenario:

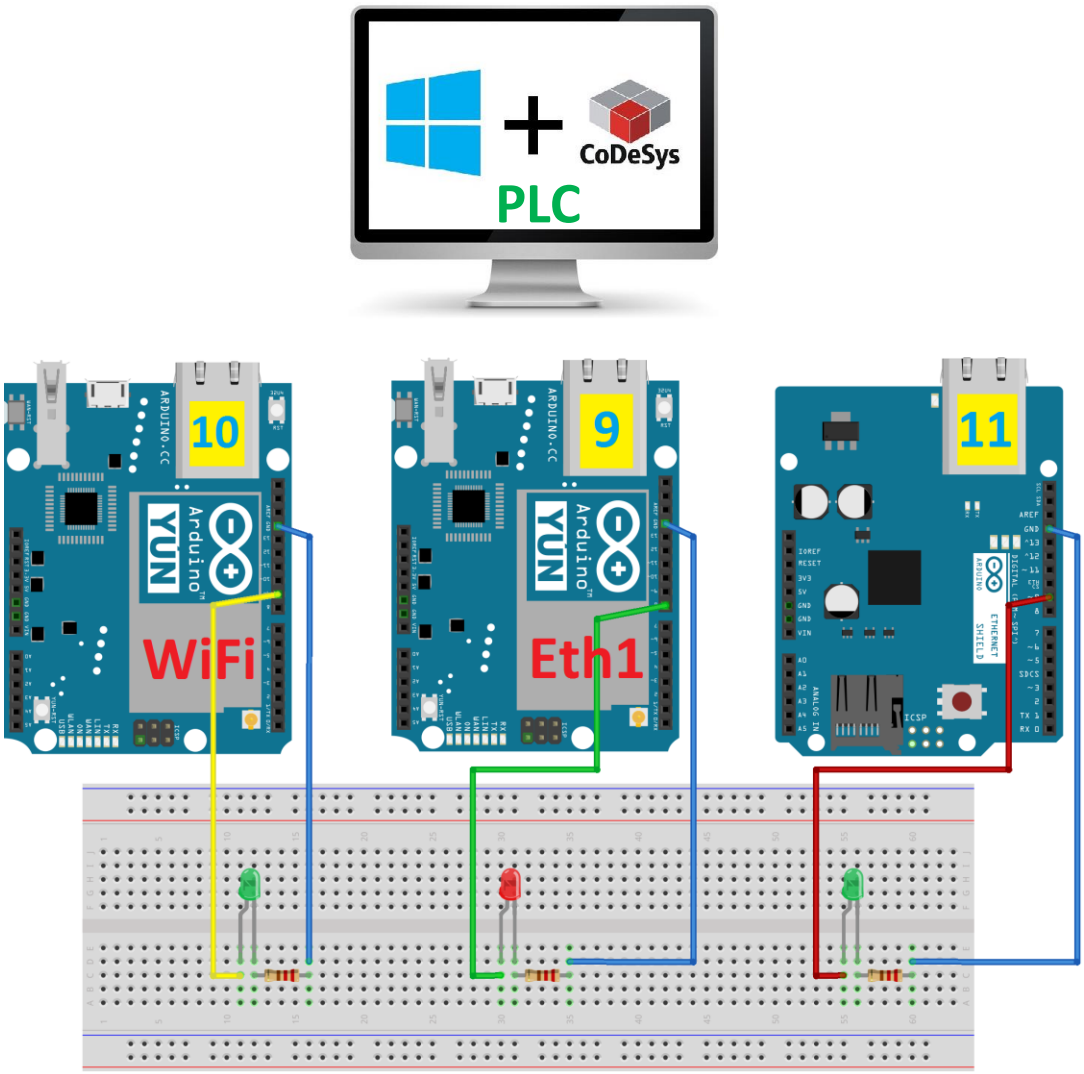

Figura 25 – Escenario de prueba de comunicación Modbus entre Arduinos y PLC virtual sobre Windows

Un punto interesante a destacar de esta tercera prueba es la peculiar característica y funcionalidad que poseen los Arduinos Yún, la cual depende según tenga el cable Ethernet conectado o no. Concretando un poco más, si el cable Ethernet está conectado, Arduino Yún se inicia tomando la interfaz EthX, en cambio, si el cable Ethernet está desconectado, se inicia tomando la interfaz WiFi. Inicialmente se pensó en la necesidad de usar una librería Modbus específica según la interfaz a usar, pero tras descubrir esto se observa que la misma librería Modbus sirve para ambas interfaces y que podemos usar en cualquier momento el Arduino Yún según nos convenga en un momento determinado sin tener que realizar ningún ajuste, simplemente decidir si usar un cable Ethernet o no.

Esto último, junto con las características y funcionalidades que facilita el entorno gráfico de Codesys, abre aún más un abanico de aplicaciones interesantes para realizar y de mucha mayor envergadura, incluso se reduce los costes de instalación gracias a las comunicaciones inalámbricas. Pero esos costes se pueden reducir aún más si dicha tarea periódica programada la realiza un equipo el cual tiene unas dimensiones reducidas, un microprocesador capaz de cumplir con creces esa tarea, y con un consumo minúsculo comparado con un ordenador. Todas esas características las recoge una Raspberry Pi.

5.4. Comunicación entre Codesys - PLC en Raspberry Pi - Arduinos

Esta cuarta prueba, es muy similar a la prueba anterior, pero con unos pequeños cambios, ya que en este caso la función de PLC Maestro la realiza una Raspberry Pi y no un PLC virtual sobre Windows como en el caso anterior.

Los esclavos seguirán siendo tres Arduinos, uno de ellos Arduino UNO y los otros dos Arduinos Yún, comunicándose estos dos últimos de forma distinta con el maestro, uno a través de Ethernet y el otro por WiFi.

El escenario continúa siendo también el mismo, cada Arduino tiene conectado un led que será sobre el que actúa el maestro. La forma correcta de conectar un led en Arduino es colocando en serie con el terminal largo (ánodo) una resistencia de al menos 220 ohmios y ésta a un pin digital de Arduino, y el otro terminal corto (cátodo) del led a tierra, pin GND. En la siguiente figura se muestra cómo debe quedar el escenario:

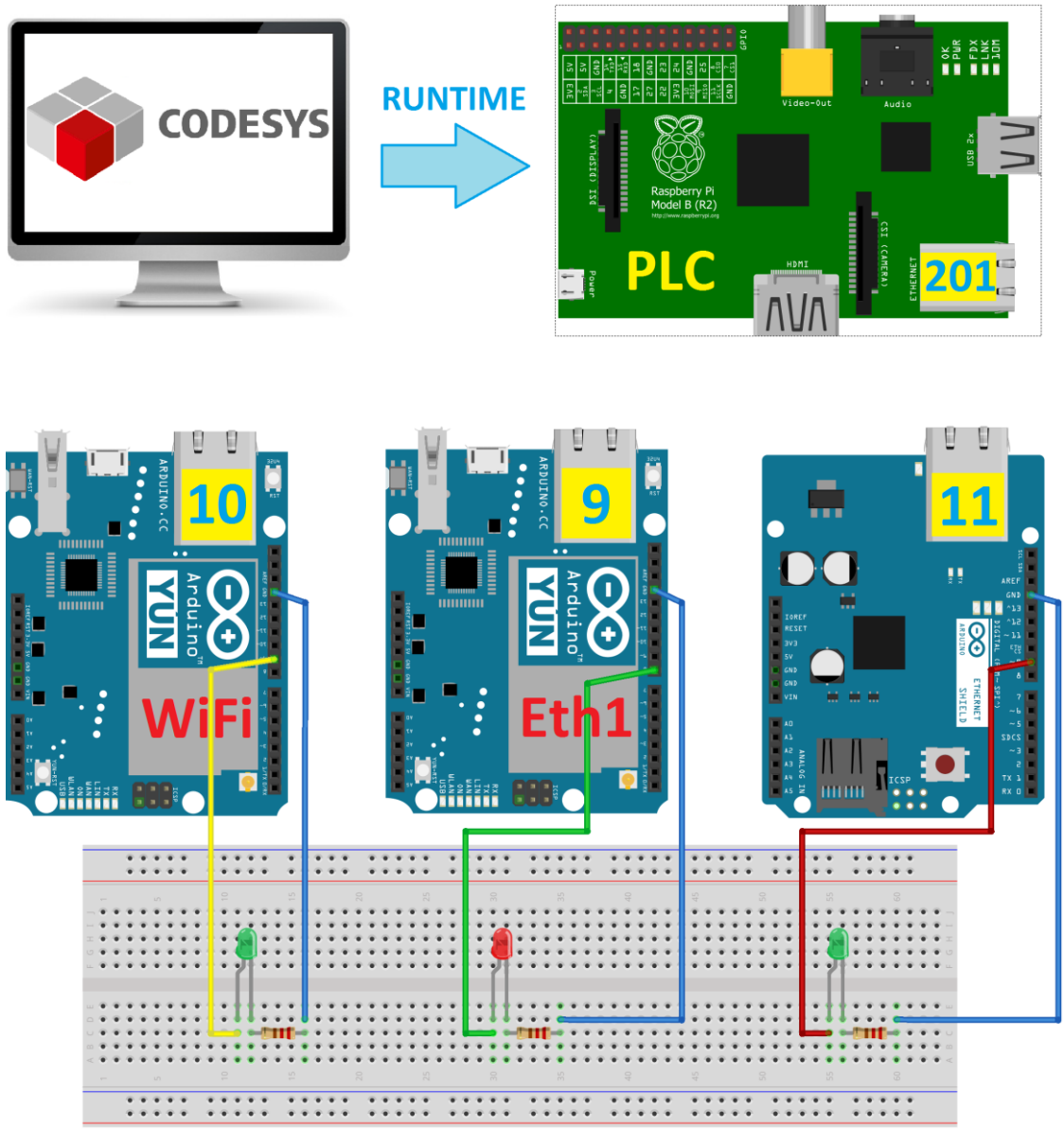

Figura 26 - Escenario de prueba de comunicación Modbus entre Arduinos y Raspberry Pi como PLC

Como se puede observar por el recorrido de pruebas funcionales que se han realizado, esta última cumple perfectamente como una plataforma de automatización de bajo coste, y cuya instalación, implementación y mantenimiento resulta sencillo.

6 ESCENARIO AVANZADO

En el capítulo anterior se presentaron distintas configuraciones posibles, llegando a la conclusión de que, para una plataforma de automatización de bajo coste, la configuración más óptima y funcional es usando el software Codesys, el cual permite configurar el escenario y los dispositivos de una manera más gráfica e intuitiva, junto con una Raspberry Pi, la cual es la encargada de procesar las funciones programadas en Codesys, manteniendo el sistema activo sin necesidad de tener este software activo sobre Windows.

Para dicha demostración se usó un circuito muy básico, el cual simplemente se encargaba de controlar dos leds. En este apartado, para demostrar el potencial de funcionamiento y versatilidad que tiene el uso de Codesys junto con Raspberry Pi y Arduinos, todos ellos comunicados a través del protocolo de comunicaciones Modbus, se ha diseñado un escenario más complejo y con distintas aplicaciones implementadas en los diferentes esclavos y gestionadas por el maestro.

En este escenario se hace uso de los dispositivos descritos y usados para las distintas pruebas de funcionalidad realizadas en el punto 5. Concretamente, estos dispositivos son una Raspberry Pi 3, un Arduino UNO con un Shield Ethernet, y dos Arduinos YÚN, todos ellos con direcciones IP fijadas para poder tener un mayor control de los dispositivos.

Se ha meditado sobre una aplicación de mayor potencial en la que todos los dispositivos interactúen entre ellos, llegando a la conclusión de que una aplicación de ese nivel puede ser el conjunto de pequeñas aplicaciones todas juntas como, por ejemplo:

- Control de un Led RGB mediante un pulsador alojado en cada uno de los esclavos.
- Obtención de la temperatura ambiente en un esclavo, y reacción de ésta de manera visual sobre otro esclavo distinto.
- Obtención de la luminosidad en un esclavo y reacción de ésta de manera sonora sobre otro esclavo distinto.

De los dispositivos nombrados anteriormente, Raspberry Pi actuaría como maestro, y los tres Arduinos actuarían como esclavos.

Para detallar y entender un poco más las funcionalidades que tienen los dispositivos según las aplicaciones antes mencionadas, se ha montado la siguiente tabla:

Dispositivo	Raspberry Pi 3	Arduino UNO	Arduino YÚN 9	Arduino YÚN 10
Patrón de diseño	Maestro y PLC	Esclavo	Esclavo	Esclavo
Dirección IP	192.168.1.**201**	192.168.1.**11**	192.168.1.**9**	192.168.1.**10**
Conectividad	Ethernet	Ethernet	WiFi	WiFi
Funcionalidad saliente	· Tx datos para procesarlos · Pantalla de explotación	· Pulsador para activar el color **B** del RG**B**	· Pulsador para activar el color **R** del **R**GB · Medida de la Temperatura	· Pulsador para activar el color **G** del R**G**B · Medida Lumínica
Funcionalidad entrante	· Rx datos para procesarlos · Logística de procesamiento y ejecución para cada 'slave'	· Según la combinación de los pulsadores se actúa sobre el Led RGB	· Según la luminosidad obtenida del sensor de luz, se activa o no la alarma sonora	· Según el rango de temperatura, se enciende uno, dos o tres leds indicando el rango visualmente.

Tabla 7 – Funcionalidades de los dispositivos según las aplicaciones

La comunicación entre los dispositivos se va a realizar a través del protocolo de comunicación Modbus TCP/IP, el cual permite la implementación sobre redes Ethernet e incluso Wireless, por lo que aumenta el grado de conectividad. Tanto Raspberry Pi como Arduino UNO tendrían conectividad con el router a través de Ethernet, y los Arduinos YÚN tendrían conectividad inalámbrica, WiFi. Se han asignado estos tipos de conectividad de manera aleatoria, siendo posible elegir en cada dispositivo la otra opción no usada. El protocolo de comunicaciones Modbus no entiende de Ethernet o WiFi, toma a los dos por igual, por tanto, es posible realizar dicho cambio en cualquier momento sin que éste afecte al sistema.

En nuestro caso, haremos uso de los "*registers*" y "*coils*". La diferencia entre ellos es que "*coils*" es usado para trabajar con valores digitales, es decir, valores booleanos, de tamaño bit, en cambio, "*registers*" es usado para trabajar con valores analógicos, los cuales tienen un amplio rango de valores que mostrar y no solamente 0 ó 1 como "*coils*". Para ello, cada "*register*" es un WORD (2 bytes = 16 bits), por tanto, el valor máximo a contemplar es el (2^16)-1 = 65535.

La siguiente tabla muestra la asignación de los "*registers*" y "*coils*" para cada dispositivo, y en ella se hace uso de la siguiente nomenclatura para referirnos a cada uno de ellos:

- Mb.R[x] ← Registro x
- Mb.C[x] ← Coil x

ARDUINO UNO				
		Valor	Lectura PLC	Escritura PLC
Coils	Mb.C[0]	Pulsador Rojo (Rx)	-	%QX0.0
	Mb.C[1]	Pulsador Verde (Rx)	-	%QX0.1
	Mb.C[2]	Pulsador Azul (Rx)	-	%QX0.0
	Mb.C[3]	Pulsador Azul (Tx)	%IX0.3	-
	Mb.C[4]	PulAzul WEB* (Rx)	-	%QX0.4
Registers	-	-	-	-

** PulAzul WEB es el pulsador de la pantalla de explotación HMI.*

Tabla 8 - Asignación de "Registers" y "Coils" en Arduino UNO

ARDUINO YUN 9				
		Valor	Lectura PLC	Escritura PLC
Coils	Mb.C[0]	Lamp1_temp (Rx)	-	%QX18.0
	Mb.C[1]	Lamp2_temp (Rx)	-	%QX18.1
	Mb.C[2]	Lamp3_temp (Rx)	-	%QX18.3
	Mb.C[3]	Pulsador Verde (Tx)	%IX18.3	-
	Mb.C[4]	PulVerde WEB (Rx)	-	%QX18.4
Registers	Mb.R[0]	Valor Sensor Luz (Tx)	%IW10	-

** PulVerde WEB es el pulsador de la pantalla de explotación HMI.*

Tabla 9 - Asignación de "Registers" y "Coils" en Arduino YÚN 9

ARDUINO YUN10				
		Valor	Lectura PLC	Escritura PLC
Coils	Mb.C[0]	Buzzer (Rx)	-	%QX36.0
	Mb.C[1]	-	-	-
	Mb.C[2]	-	-	-
	Mb.C[3]	Pulsador Rojo (Tx)	%IX36.3	-
	Mb.C[4]	PulRojo WEB (Rx)	-	%QX36.4
Registers	Mb.R[0]	Parte entera Temperatura (Tx)	%IW19	-
	Mb.R[1]	Parte decimal Temperatura (Tx)	%IW20	-

** PulRojo WEB es el pulsador de la pantalla de explotación HMI.*

Tabla 10 - Asignación de "Registers" y "Coils" en Arduino YÚN 10

Para poder visualizar y controlar todos los valores tanto de los pulsadores como de los sensores se hacen pasar por el PLC de manera que éste se encargue de procesarlos, en el caso de que hiciese falta, y redirigirlos hacía el dispositivo sobre el que tiene que actuar. Además, como el PLC es el encargado de mostrar una pantalla de explotación, necesita de estos valores para mostrar la información adecuada.

Como se comentó anteriormente, el escenario para estas aplicaciones es más complejo, cada Arduino tiene una funcionalidad determinada como se recoge en la tabla 8. En la siguiente figura se muestra cómo debe quedar el escenario:

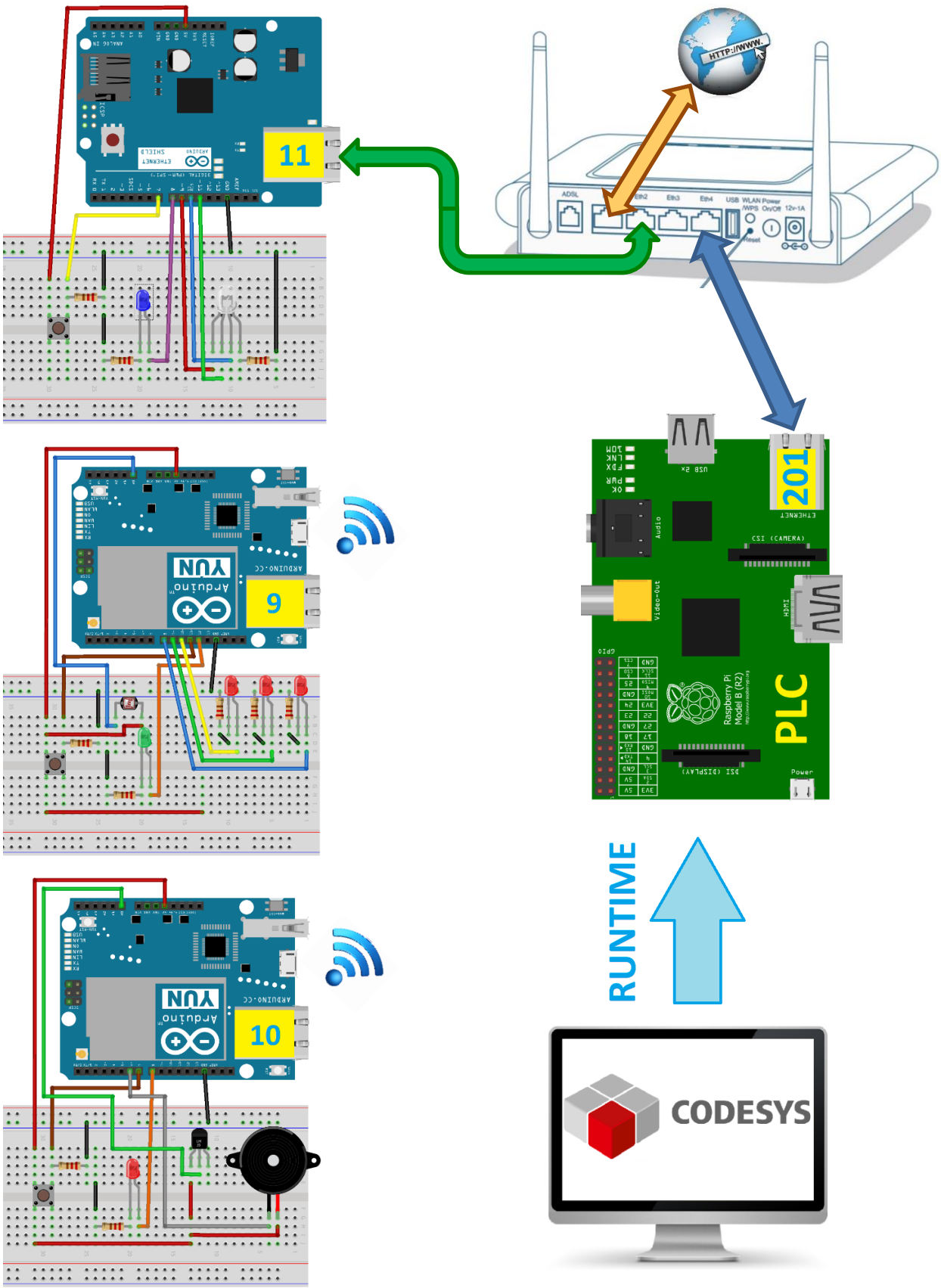

Figura 27 – Escenario Avanzado

Tras comprobar su correcto funcionamiento, se diseña una pantalla de explotación. En este caso, el diseño usado es el que se muestra en la siguiente figura:

Figura 28 – Pantalla de explotación creada en Codesys

Para visualizar la pantalla de explotación es necesario acceder desde cualquier navegador a la siguiente dirección web, cambiando únicamente "IPraspberry" por la dirección IP de la Raspberry Pi:

http://IPraspberry:8080/webvisu.htm

De esta manera, solo tenemos la pantalla de explotación diseñada, pero si además queremos visualizar en directo como se encuentra el sistema, o incluso ver la actuación sobre el sistema tras manipular la pantalla de explotación en directo, se ha implementado una página web en la que se accederá según estemos dentro o fuera de la red local:

Figura 29 – Pantalla principal de la Web diseñada para HMI

Una vez que accedemos por una de las dos opciones, la web que se nos muestra es la siguiente:

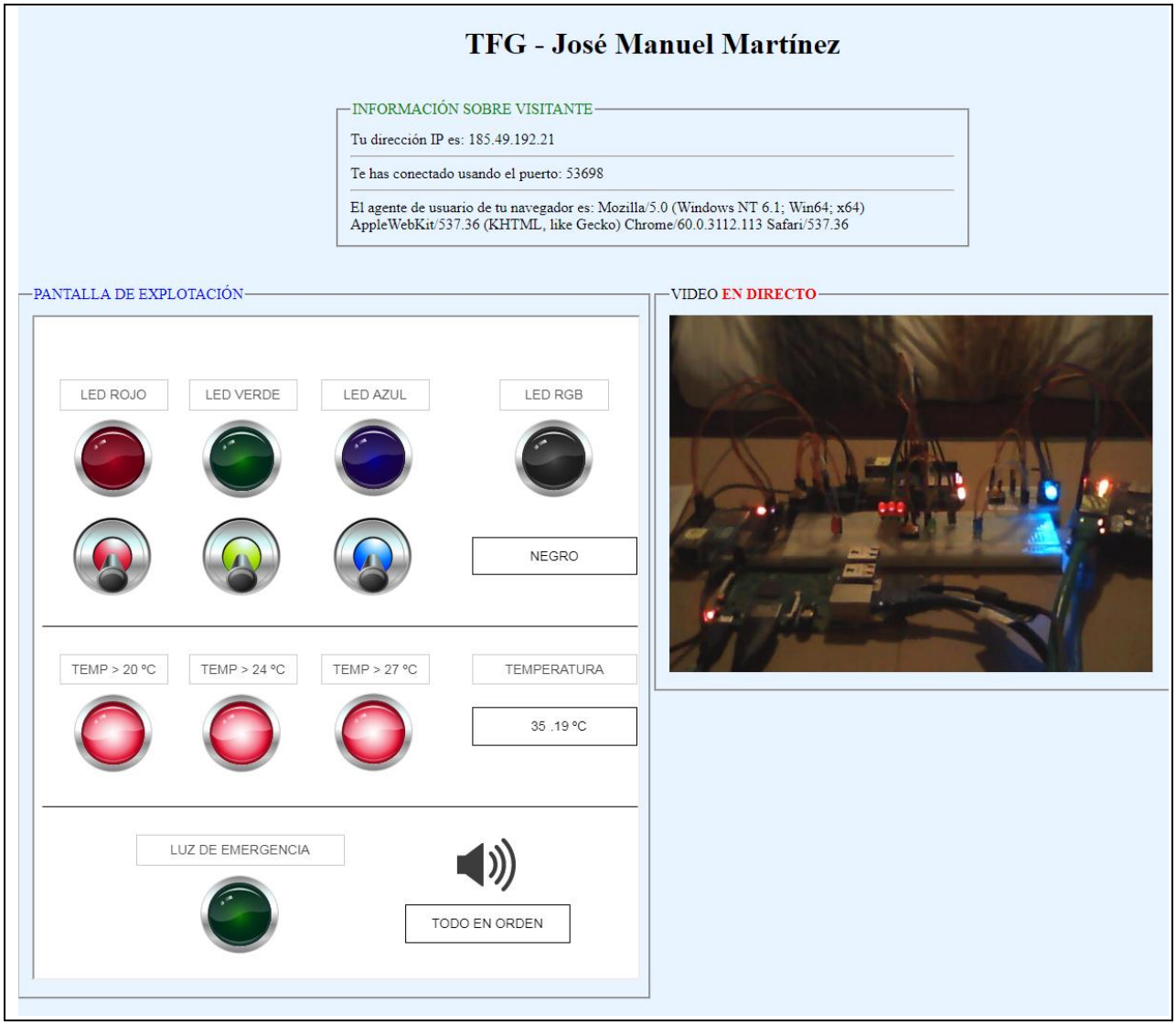

Figura 30 – Web implementada con la pantalla de explotación y vídeo en directo

7 CONCLUSIONES Y LÍNEAS FUTURAS DE TRABAJO

7.1. Conclusiones

En el desarrollo de este documento ha quedado reflejada la gran utilidad que tiene una plataforma modular basada en procesadores ARM como alternativa a un PLC en la automatización de procesos. En nuestro caso, el dispositivo base de la plataforma es Raspberry Pi, que junto con el software Codesys, constituyen una plataforma de automatización de bajo coste, siempre siguiendo el estándar internacional IEC 61131-3.

Dicha plataforma se basa en el uso del software libre, que unido al protocolo de comunicaciones Modbus TCP/IP bien por vía Ethernet o de forma inalámbrica, abre las puertas a la introducción de otros dispositivos en el proyecto como son Arduino UNO y Arduino Yún. Estos dispositivos amplían las capacidades de la plataforma y refuerzan el concepto de *Internet of Things (IoT)* dentro del proyecto.

Otra gran ventaja derivada del empleo de software libre es la posibilidad de implementar otros protocolos de comunicación propios de *IoT*. La libertad en cuanto a uso de hardware y software en la plataforma de automatización refuerza el carácter modular de ésta.

El futuro de la automatización industrial pasa por el uso de plataformas modulares parecidas a la diseñada en este proyecto, y estarán constituidas por componentes software y hardware abiertos, pudiéndose integrar en el concepto *Industry 4.0*.

También sería de gran utilidad en el campo de la domótica o en el de la docencia, ya que al ser una plataforma de automatización basada en Raspberry Pi, puede servir para enseñar programación basada en el estándar IEC 61131-3 a los alumnos de forma bastante económica.

7.2. Líneas futuras de trabajo

Partiendo de esta base, las líneas de trabajo que se pueden seguir son:

- Añadir al proyecto el uso de dispositivos de dimensiones más reducidas e igualmente programables a un Arduino como son los llamados ESP8266. Son dispositivos que están en gran auge y que tendremos que tener muy en cuenta en el desarrollo IoT, IIoT y la Industria 4.0. Todo gracias a su reducido coste, su bajo consumo, y cada vez más, su facilidad de uso.
- Expandir una red paralela junto a la ya instalada con autómatas profesionales en el laboratorio e investigar las ventajas y desventajas, rendimiento, versatilidad, etc, de un autómata de bajo coste basado en Raspberry Pi frente a un autómata profesional.
- Implementar un software que permita gestionar una red de autómatas de bajo coste desde un único puesto, tanto de manera local como remotamente desde cualquier lugar. Un ejemplo de software con esta funcionalidad es *remot3.it*.

REFERENCIAS

[1] National Instruments, «Información sobre el Protocolo Modbus,» 16 October 2014. [En línea]. Available: http://www.ni.com/white-paper/52134/es/.

[2] Modbus-IDA, «Modbus Application Protocol Specification V.1.1b,» 28 December 2006. [En línea]. Available: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf.

[3] Modbus-IDA, «Modelo de datos,» [En línea]. Available: https://es.wikipedia.org/wiki/Modbus.

[4] A. Barragan, «Modbus TCP/IP,» [En línea]. Available: http://uhu.es/antonio.barragan/content/modbus-tcp.

[5] «Raspberry Pi,» [En línea]. Available: http://www.raspberrypi.org/.

[6] «Tabla técnica Raspberry Pi,» Febrero 2017. [En línea]. Available: https://raspberryparatorpes.net/raspberry-pi-tabla-tecnica-completa/.

[7] «Codesys,» [En línea]. Available: https://www.codesys.com/.

[8] «Arduino,» [En línea]. Available: http://www.arduino.cc/es/.

[9] «Wiring,» [En línea]. Available: http://wiring.org.co/.

[10] «Processing,» [En línea]. Available: http://www.processing.org/.

[11] «Especificaciones Arduino UNO,» Mayo 2013. [En línea]. Available: http://www3.gobiernodecanarias.org/medusa/ecoblog/ralvgon/files/2013/05/Caracter%C3%ADsticas-Arduino.pdf.

[12] Aprendiendo Arduino, «Ethernet Shield,» [En línea]. Available: https://aprendiendoarduino.wordpress.com/2016/07/04/ethernet-shield/.

[13] «Ethernet / Ethernet 2 library,» [En línea]. Available: http://arduino.cc/en/Reference/Ethernet.

[14] Arduino, «Bridge Library for Yún devices,» [En línea]. Available: https://www.arduino.cc/en/Reference/YunBridgeLibrary.

[15] A. G. González, «Arduino Yún: Características y Capacidades,» 30 Mayo 2014. [En línea]. Available: http://panamahitek.com/arduino-yun-caracteristicas-y-capacidades/.

[16] Observa Telecom, «Manual de Usuario,» [En línea]. Available:

http://www.movistar.es/rpmm/estaticos/residencial/fijo/banda-ancha-adsl/manuales/modem-router-inalambricos-adsl/home-station-adsl-observa-rta/manual-usuario-fabricante.pdf.

[17] Codesys, «Manual Installation and Start,» [En línea]. Available: http://support.crosscontrol.com/sites/default/files/documentation/Software/CoDeSys/Documentation/CODESYS%20Installation%20and%20Start.pdf.

[18] comohacer.eu, «Control Arduino por USB,» [En línea]. Available: https://comohacer.eu/control-de-arduino-mediante-la-raspberry-pi/.

[19] «Remote Manage Networked Devices Anywhere,» [En línea]. Available: https://www.remot3.it.

[20] «Modbus Library for the Arduino,» [En línea]. Available: http://myarduinoprojects.com/modbus.html.

[21] J. Bartolomé, «El protocolo MODBUS,» 2011. [En línea]. Available: http://www.tolaemon.com/docs/modbus.htm#campos_de_la_tram.

[22] Raspberry Pi, «SO Raspbian,» [En línea]. Available: https://www.raspberrypi.org/downloads/.

[23] «Configurar IP estática Raspberry,» [En línea]. Available: [https://raspberryparatorpes.net/instalacion/poner-la-direccin-ip-fija-en-raspbian/.

[24] «Servidor SAMBA,» [En línea]. Available: https://raspberryparatorpes.net/proyectos/instalar-samba-preparando-un-nas-o-servidor-casero-3/.

GLOSARIO

PLC	Programmable Logic Controller
SBC	Single Board Computer
CPS	Cyber-Physical Systems
IoT	Internet of Things
IIoT	Industrial Internet of Things
CPPS	Cyber-Physical Production Systems
OSI	Open System Interconnection
RTU	Arquitectura Maestro-Esclavo
ASCII	American Standard Code for Information Interchange
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IP	Internet Protocol
IETF	Internet Engineering Task Force
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
I2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
CLI	Command Line Interface
POUs	Unidades de Organización de Programa
IDE	Integrated Development Environment
SSH	Secure Shell
RGB	red, green, blue

ANEXOS

Anexo I: Librería Modbus para Arduino

Esta librería es necesaria si se quiere hacer uso del protocolo Modbus entre placas Arduinos y está compuesta por: *MgsModbus.h* y por *MgsModbus.cpp*, siendo este último quien recoge el conjunto de funciones que utiliza el protocolo Modbus.

- *MgsModbus.h*

```
/*
  MgsModbus.h - an Arduino library for a Modbus TCP master and slave.
  V-0.1.1 Copyright (C) 2013  Marco Gerritse
  written and tested with Arduino 1.0

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

  For this library the following library is used as start point:

    [1] Mudbus.h - an Arduino library for a Modbus TCP slave.
        Copyright (C) 2011  Dee Wykoff
    [2] Function codes 15 & 16 by Martin Pettersson

  The following references are used to write this library:

    [3] Open Modbus/Tcp specification, Release 1.0, 29 March 1999
        By Andy Swales, Schneider Electric
    [4] Modbus application protocol specification V1.1b3, 26 april 202
        From http:/www.modbus.org

  External software used for testing:

    [5] modpoll - www.modbusdriver.com/modpoll.html
```

```
    [6] ananas - www.tuomio.fi/ananas
    [7] mod_rssim - www.plcsimulator.org
    [8] modbus master - www.cableone.net/mblansett/

  This library use a single block of memory for all modbus data (mbData[]
array). The
  same data can be reached via several modbus functions, either via a 16 bit
access
  or via an access bit. The length of MbData must at least 1.

  For the master the following modbus functions are implemented: 1, 2, 3, 4,
5, 6, 15, 16
  For the slave the following modbus functions are implemented: 1, 2, 3, 4,
5, 6, 15, 16

  The internal and external addresses are 0 (zero) based


  V-0.1.1 2013-06-02
  bugfix

  V-0.1.0 2013-03-02
  initinal version
*/


#include "Arduino.h"

#include <SPI.h>
#include <Ethernet.h>

#ifndef MgsModbus_h
#define MgsModbus_h

#define MbDataLen 30 // length of the MdData array
#define MB_PORT 502

enum MB_FC {
  MB_FC_NONE                    = 0,
  MB_FC_READ_COILS              = 1,
  MB_FC_READ_DISCRETE_INPUT     = 2,
  MB_FC_READ_REGISTERS          = 3,
  MB_FC_READ_INPUT_REGISTER     = 4,
  MB_FC_WRITE_COIL              = 5,
  MB_FC_WRITE_REGISTER          = 6,
  MB_FC_WRITE_MULTIPLE_COILS    = 15,
  MB_FC_WRITE_MULTIPLE_REGISTERS = 16
};

class MgsModbus
{
public:
  // general
  MgsModbus();
  word MbData[MbDataLen]; // memory block that holds all the modbus user data
  boolean GetBit(word Number);
  boolean SetBit(word Number,boolean Data); // returns true when the number
is in the MbData range
  // modbus master
  void Req(MB_FC FC, word Ref, word Count, word Pos);
  void MbmRun();
  IPAddress remSlaveIP;
```

```
  // modbus slave
  void MbsRun();
  word GetDataLen();
private:
  // general
  MB_FC SetFC(int fc);
  // modbus master
  uint8_t MbmByteArray[260]; // send and recieve buffer
  MB_FC MbmFC;
  int MbmCounter;
  void MbmProcess();
  word MbmPos;
  word MbmBitCount;
  //modbus slave
  uint8_t MbsByteArray[260]; // send and recieve buffer
  MB_FC MbsFC;
};

#endif
```

- *MgsModbus.cpp*

```
#include "MgsModbus.h"

// For Arduino 1.0
EthernetServer MbServer(MB_PORT);
EthernetClient MbmClient;

// #define DEBUG

MgsModbus::MgsModbus()
{
}


//****************** Send data for ModBusMaster ****************
void MgsModbus::Req(MB_FC FC, word Ref, word Count, word Pos)
{
  MbmFC = FC;
  byte ServerIp[] = {192,168,1,10};
  MbmByteArray[0] = 0;  // ID high byte
  MbmByteArray[1] = 1;  // ID low byte
  MbmByteArray[2] = 0;  // protocol high byte
  MbmByteArray[3] = 0;  // protocol low byte
  MbmByteArray[5] = 6;  // Lenght low byte;
  MbmByteArray[4] = 0;  // Lenght high byte
  MbmByteArray[6] = 1;  // unit ID
  MbmByteArray[7] = FC; // function code
  MbmByteArray[8] = highByte(Ref);
  MbmByteArray[9] = lowByte(Ref);
  //****************** Read Coils (1) & Read Input discretes (2)
**********************
  if(FC == MB_FC_READ_COILS || FC == MB_FC_READ_DISCRETE_INPUT) {
    if (Count < 1) {Count = 1;}
    if (Count > 125) {Count = 2000;}
    MbmByteArray[10] = highByte(Count);
    MbmByteArray[11] = lowByte(Count);
  }
  //****************** Read Registers (3) & Read Input registers (4)
******************
```

```
  if(FC == MB_FC_READ_REGISTERS || FC == MB_FC_READ_INPUT_REGISTER) {
    if (Count < 1) {Count = 1;}
    if (Count > 125) {Count = 125;}
    MbmByteArray[10] = highByte(Count);
    MbmByteArray[11] = lowByte(Count);
  }
  //****************** Write Coil (5) **********************
  if(MbmFC == MB_FC_WRITE_COIL) {
    if (GetBit(Pos)) {MbmByteArray[10] = 0xFF;} else {MbmByteArray[10] = 0;}
// 0xFF coil on 0x00 coil off
    MbmByteArray[11] = 0; // always zero
  }
  //****************** Write Register (6) ******************
  if(MbmFC == MB_FC_WRITE_REGISTER) {
    MbmByteArray[10] = highByte(MbData[Pos]);
    MbmByteArray[11] = lowByte(MbData[Pos]);
  }
  //****************** Write Multiple Coils (15) **********************
  // not fuly tested
  if(MbmFC == MB_FC_WRITE_MULTIPLE_COILS) {
    if (Count < 1) {Count = 1;}
    if (Count > 800) {Count = 800;}
    MbmByteArray[10] = highByte(Count);
    MbmByteArray[11] = lowByte(Count);
    MbmByteArray[12] = (Count + 7) /8;
    MbmByteArray[4] = highByte(MbmByteArray[12] + 7); // Lenght high byte
    MbmByteArray[5] = lowByte(MbmByteArray[12] + 7); // Lenght low byte;
    for (int i=0; i<Count; i++) {
      bitWrite(MbmByteArray[13+(i/8)],i-((i/8)*8),GetBit(Pos+i));
    }
  }
  //****************** Write Multiple Registers (16) ******************
  if(MbmFC == MB_FC_WRITE_MULTIPLE_REGISTERS) {
    if (Count < 1) {Count = 1;}
    if (Count > 100) {Count = 100;}
    MbmByteArray[10] = highByte(Count);
    MbmByteArray[11] = lowByte(Count);
    MbmByteArray[12] = (Count*2);
    MbmByteArray[4] = highByte(MbmByteArray[12] + 7); // Lenght high byte
    MbmByteArray[5] = lowByte(MbmByteArray[12] + 7); // Lenght low byte;
    for (int i=0; i<Count;i++) {
      MbmByteArray[(i*2)+13] = highByte (MbData[Pos + i]);
      MbmByteArray[(i*2)+14] = lowByte (MbData[Pos + i]);
    }
  }
  //****************** ?? ******************
  if (MbmClient.connect(ServerIp,502)) {
    #ifdef DEBUG
      Serial.println("connected with modbus slave");
      Serial.print("Master request: ");
      for(int i=0;i<MbmByteArray[5]+6;i++) {
        if(MbmByteArray[i] < 16){Serial.print("0");}
        Serial.print(MbmByteArray[i],HEX);
        if (i != MbmByteArray[5]+5) {Serial.print(".");} else
{Serial.println();}
      }
    #endif
    for(int i=0;i<MbmByteArray[5]+6;i++) {
      MbmClient.write(MbmByteArray[i]);
    }
    MbmCounter = 0;
    MbmByteArray[7] = 0;
```

```
    MbmPos = Pos;
    MbmBitCount = Count;
  } else {
    #ifdef DEBUG
      Serial.println("connection with modbus master failed");
    #endif
    MbmClient.stop();
  }
}


//****************** Recieve data for ModBusMaster ***************
void MgsModbus::MbmRun()
{
  //****************** Read from socket ***************
  while (MbmClient.available()) {
    MbmByteArray[MbmCounter] = MbmClient.read();
    if (MbmCounter > 4)  {
      if (MbmCounter == MbmByteArray[5] + 5) { // the full answer is recieved
        MbmClient.stop();
        MbmProcess();
        #ifdef DEBUG
          Serial.println("recieve klaar");
        #endif
      }
    }
    MbmCounter++;
  }
}

void MgsModbus::MbmProcess()
{
  MbmFC = SetFC(int (MbmByteArray[7]));
  #ifdef DEBUG
    for (int i=0;i<MbmByteArray[5]+6;i++) {
      if(MbmByteArray[i] < 16) {Serial.print("0");}
      Serial.print(MbmByteArray[i],HEX);
      if (i != MbmByteArray[5]+5) {Serial.print(".");
      } else {Serial.println();}
    }
  #endif
  //****************** Read Coils (1) & Read Input discretes (2)
**********************
  if(MbmFC == MB_FC_READ_COILS || MbmFC == MB_FC_READ_DISCRETE_INPUT) {
    word Count = MbmByteArray[8] * 8;
    if (MbmBitCount < Count) {
      Count = MbmBitCount;
    }
    for (int i=0;i<Count;i++) {
      if (i + MbmPos < MbDataLen * 16) {
        SetBit(i + MbmPos,bitRead(MbmByteArray[(i/8)+9],i-((i/8)*8)));
      }
    }
  }
  //****************** Read Registers (3) & Read Input registers (4)
******************
  if(MbmFC == MB_FC_READ_REGISTERS || MbmFC == MB_FC_READ_INPUT_REGISTER) {
    word Pos = MbmPos;
    for (int i=0;i<MbmByteArray[8];i=i+2) {
      if (Pos < MbDataLen) {
        MbData[Pos] = (MbmByteArray[i+9] * 0x100) + MbmByteArray[i+1+9];
        Pos++;
      }
```

```
    }
  }
  //****************** Write Coil (5) **********************
  if(MbmFC == MB_FC_WRITE_COIL){
  }
  //****************** Write Register (6) ******************
  if(MbmFC == MB_FC_WRITE_REGISTER){
  }
  //****************** Write Multiple Coils (15) ***********************
  if(MbmFC == MB_FC_WRITE_MULTIPLE_COILS){
  }
  //****************** Write Multiple Registers (16) ******************
  if(MbmFC == MB_FC_WRITE_MULTIPLE_REGISTERS){
  }
}


//****************** Recieve data for ModBusSlave ***************
void MgsModbus::MbsRun()
{
  //****************** Read from socket ***************
  EthernetClient client = MbServer.available();
  if(client.available())
  {
    delay(10);
    int i = 0;
    while(client.available())
    {
      MbsByteArray[i] = client.read();
      i++;
    }
    MbsFC = SetFC(MbsByteArray[7]);  //Byte 7 of request is FC
  }
  int Start, WordDataLength, ByteDataLength, CoilDataLength, MessageLength;
  //****************** Read Coils (1 & 2) **********************
  if(MbsFC == MB_FC_READ_COILS || MbsFC == MB_FC_READ_DISCRETE_INPUT) {
    Start = word(MbsByteArray[8],MbsByteArray[9]);
    CoilDataLength = word(MbsByteArray[10],MbsByteArray[11]);
    ByteDataLength = CoilDataLength / 8;
    if(ByteDataLength * 8 < CoilDataLength) ByteDataLength++;
    CoilDataLength = ByteDataLength * 8;
    MbsByteArray[5] = ByteDataLength + 3; //Number of bytes after this one.
    MbsByteArray[8] = ByteDataLength;     //Number of bytes after this one
(or number of bytes of data).
    for(int i = 0; i < ByteDataLength ; i++)
    {
      MbsByteArray[9 + i] = 0; // To get all remaining not written bits zero
      for(int j = 0; j < 8; j++)
      {
        bitWrite(MbsByteArray[9 + i], j, GetBit(Start + i * 8 + j));
      }
    }
    MessageLength = ByteDataLength + 9;
    client.write(MbsByteArray, MessageLength);
    MbsFC = MB_FC_NONE;
  }
  //****************** Read Registers (3 & 4) ******************
  if(MbsFC == MB_FC_READ_REGISTERS || MbsFC == MB_FC_READ_INPUT_REGISTER) {
    Start = word(MbsByteArray[8],MbsByteArray[9]);
    WordDataLength = word(MbsByteArray[10],MbsByteArray[11]);
    ByteDataLength = WordDataLength * 2;
    MbsByteArray[5] = ByteDataLength + 3; //Number of bytes after this one.
```

```
    MbsByteArray[8] = ByteDataLength;     //Number of bytes after this one
(or number of bytes of data).
    for(int i = 0; i < WordDataLength; i++)
    {
      MbsByteArray[ 9 + i * 2] = highByte(MbData[Start + i]);
      MbsByteArray[10 + i * 2] =  lowByte(MbData[Start + i]);
    }
    MessageLength = ByteDataLength + 9;
    client.write(MbsByteArray, MessageLength);
    MbsFC = MB_FC_NONE;
  }
  //****************** Write Coil (5) **********************
  if(MbsFC == MB_FC_WRITE_COIL) {
    Start = word(MbsByteArray[8],MbsByteArray[9]);
    if (word(MbsByteArray[10],MbsByteArray[11]) ==
0xFF00){SetBit(Start,true);}
    if (word(MbsByteArray[10],MbsByteArray[11]) ==
0x0000){SetBit(Start,false);}
    MbsByteArray[5] = 2; //Number of bytes after this one.
    MessageLength = 8;
    client.write(MbsByteArray, MessageLength);
    MbsFC = MB_FC_NONE;
  }
  //****************** Write Register (6) ******************
  if(MbsFC == MB_FC_WRITE_REGISTER) {
    Start = word(MbsByteArray[8],MbsByteArray[9]);
    MbData[Start] = word(MbsByteArray[10],MbsByteArray[11]);
    MbsByteArray[5] = 6; //Number of bytes after this one.
    MessageLength = 12;
    client.write(MbsByteArray, MessageLength);
    MbsFC = MB_FC_NONE;
  }
  //****************** Write Multiple Coils (15) **********************
  if(MbsFC == MB_FC_WRITE_MULTIPLE_COILS) {
    Start = word(MbsByteArray[8],MbsByteArray[9]);
    CoilDataLength = word(MbsByteArray[10],MbsByteArray[11]);
    MbsByteArray[5] = 6;
    for(int i = 0; i < CoilDataLength; i++)
    {
      SetBit(Start + i,bitRead(MbsByteArray[13 + (i/8)],i-((i/8)*8)));
    }
    MessageLength = 12;
    client.write(MbsByteArray, MessageLength);
    MbsFC = MB_FC_NONE;
  }
  //****************** Write Multiple Registers (16) ******************
  if(MbsFC == MB_FC_WRITE_MULTIPLE_REGISTERS) {
    Start = word(MbsByteArray[8],MbsByteArray[9]);
    WordDataLength = word(MbsByteArray[10],MbsByteArray[11]);
    ByteDataLength = WordDataLength * 2;
    MbsByteArray[5] = 6;
    for(int i = 0; i < WordDataLength; i++)
    {
      MbData[Start + i] =  word(MbsByteArray[ 13 + i * 2],MbsByteArray[14 + i
* 2]);
    }
    MessageLength = 12;
    client.write(MbsByteArray, MessageLength);
    MbsFC = MB_FC_NONE;
  }
}
```

```
//****************** ?? ******************
MB_FC MgsModbus::SetFC(int fc)
{
  MB_FC FC;
  FC = MB_FC_NONE;
  if(fc == 1) FC = MB_FC_READ_COILS;
  if(fc == 2) FC = MB_FC_READ_DISCRETE_INPUT;
  if(fc == 3) FC = MB_FC_READ_REGISTERS;
  if(fc == 4) FC = MB_FC_READ_INPUT_REGISTER;
  if(fc == 5) FC = MB_FC_WRITE_COIL;
  if(fc == 6) FC = MB_FC_WRITE_REGISTER;
  if(fc == 15) FC = MB_FC_WRITE_MULTIPLE_COILS;
  if(fc == 16) FC = MB_FC_WRITE_MULTIPLE_REGISTERS;
  return FC;
}


word MgsModbus::GetDataLen()
{
  return MbDataLen;
}


boolean MgsModbus::GetBit(word Number)
{
  int ArrayPos = Number / 16;
  int BitPos = Number - ArrayPos * 16;
  boolean Tmp = bitRead(MbData[ArrayPos],BitPos);
  return Tmp;
}


boolean MgsModbus::SetBit(word Number,boolean Data)
{
  int ArrayPos = Number / 16;
  int BitPos = Number - ArrayPos * 16;
  boolean Overrun = ArrayPos > MbDataLen * 16; // check for data overrun
  if (!Overrun){
    bitWrite(MbData[ArrayPos],BitPos,Data);
  }
  return Overrun;
}
```

Anexo II: Preparación del entorno en Raspberry Pi 3

El proceso de preparación del entorno en Raspberry Pi va a consistir en un conjunto de puntos con un orden determinado:

1. Instalación del sistema operativo en la tarjeta SD de Raspberry Pi

Raspberry Pi emplea Linux como sistema operativo. En la actualidad, existen múltiples distribuciones de Linux para Raspberry Pi, pero para este proyecto vamos a centrarnos en la oficial y más usada, Raspbian [22], la cual se podrá descargar generalmente en un archivo con extensión .img.

Para instalar el sistema operativo existen distintas formas de hacerlo, dependiendo del sistema operativo en el que se haga, ya sea Windows, GNU/Linux o OS X. En nuestro caso, se va a explicar el proceso de instalación sobre Windows, tal y como se indica en http://freeberries.org/2013/12/instalar-sistema-en-sd-para-raspberry-pi/.

En este sistema, el proceso de tan sencillo como copiar el archivo descargado .img a la tarjeta SD. Para hacer esto, se va a utilizar Win32 Disk Imager, una aplicación que permite grabar imágenes en memorias SD de una forma cómoda.

El proceso consiste en escoger el fichero que contenga la imagen del sistema operativo en "Image File", elegir la unidad correcta asociada a la tarjeta SD en "Device" y finalmente pulsar el botón "Write" para que se inicie la grabación de la imagen en la SD.

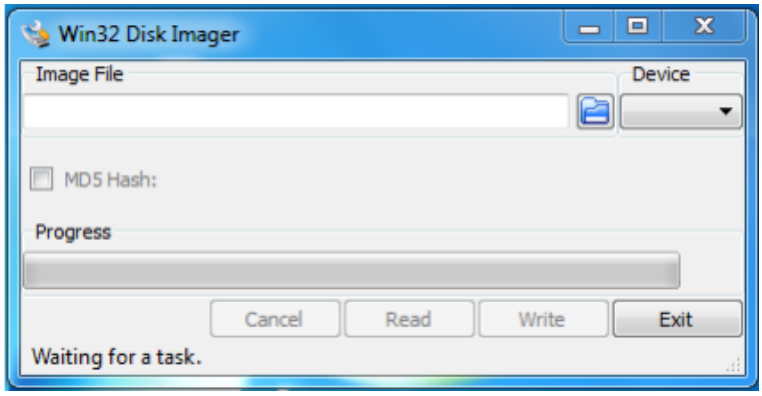

Figura 31 – Programa Win32 Disk Imager

Una vez finalizado el proceso, ya se puede insertar la tarjeta en Raspberry Pi y comenzar con la configuración.

2. Configuración inicial y de red en Raspbian

La primera vez que arrancamos tras instalar Raspbian se presenta en pantalla la herramienta "raspi-config" (ver figura 63). Esta herramienta resulta bastante útil y la usaremos más adelante para configurar nuestra Raspberry Pi para el proyecto. De momento salimos del menú seleccionando "Finish" para lanzar la interfaz gráfica de Raspbian.

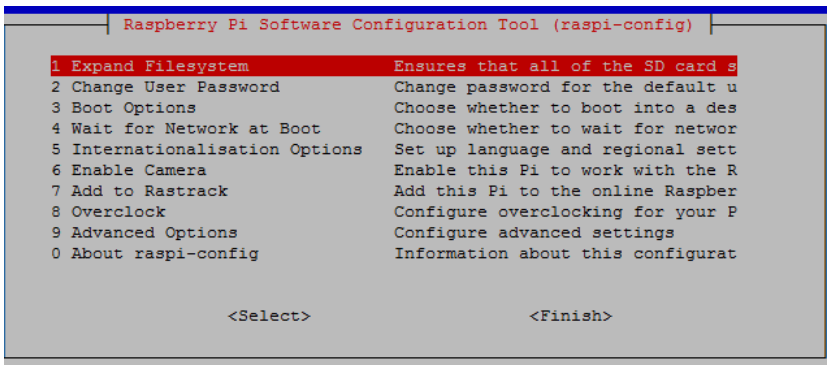

Figura 32 – Herramienta raspi-config

Una vez que nos hemos familiarizado con el escritorio de Raspbian abrimos el Terminal de Linux para configurar la Raspberry Pi para su uso en el proyecto.

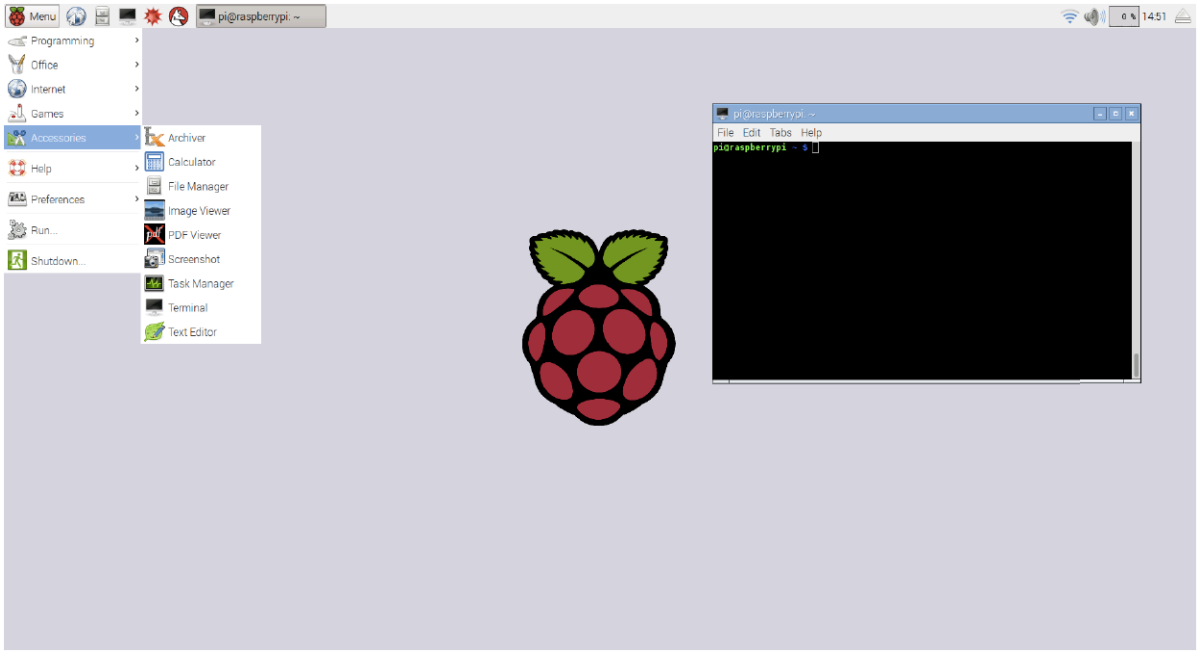

Figura 33 – Escritorio Raspbian

El primer paso para configurar la Raspberry Pi es actualizar Raspbian. Para ello se introducen los siguientes comandos en Terminal:

- `sudo apt-get update`
- `sudo apt-get upgrade`
- `sudo apt-get dist-upgrade`
- `sudo reboot`

A continuación, abrimos la herramienta "raspi-config":

- `sudo raspi-config`

Raspi-config te permite expandir el sistema de archivos aprovechando el tamaño de la SD insertada, para ello es necesario pulsar sobre la primera opción que nos muestra "Expand Filesystem" y a continuación pulsamos en "Ok".

Para poder acceder a Raspberry Pi desde un terminal SSH en nuestro PC, de forma que no sea necesario conectar Raspberry Pi a una pantalla, navegamos por el menú de raspi-config hasta "Advanced Options", y una vez ahí activamos la opción "SSH". El usuario y contraseña por defecto es "pi" y "raspberry" respectivamente.

Una vez hemos realizado estas configuraciones, reiniciamos la Raspberry Pi.

Ahora vamos a configurar una IP estática [23] ya que es necesaria para este proyecto y resulta más cómoda a la hora de manejar Raspberry Pi sin tenerla conectada a una pantalla. Por defecto, viene configurada con una IP dinámica y es el router quien le asigna una IP dentro de un rango determinado. Por tanto, para saber la IP actual de la Raspberry Pi, ejecutamos en el terminal siguiente comando (ver figura 65):

- `sudo raspi-config`

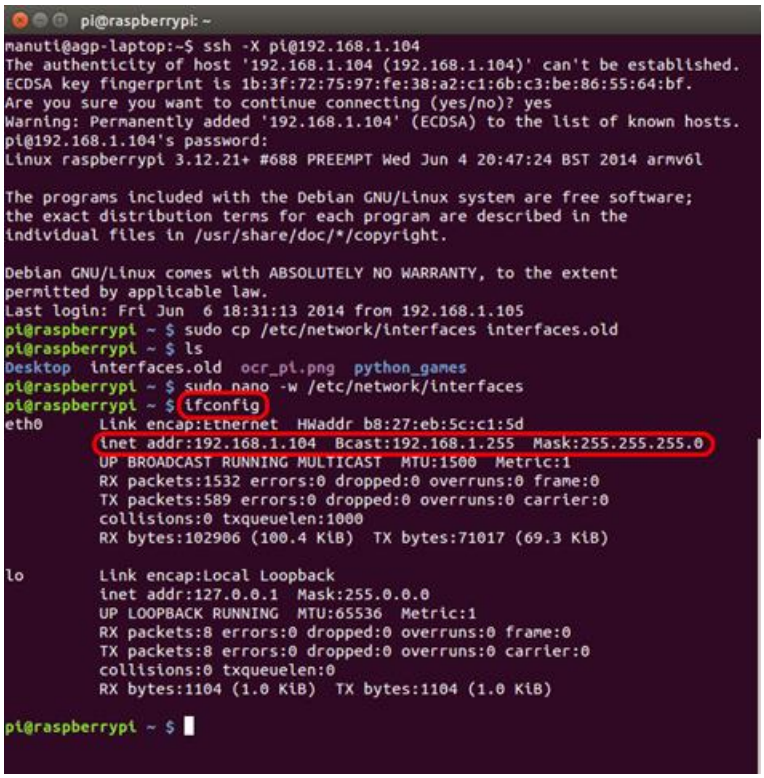

Figura 34 – Salida del comando ifconfig en Raspbian

Antes de cambiar a una dirección IP estática, hacemos una copia del fichero que vamos a cambiar con el comando:

- `sudo cp /etc/network/interfaces interfaces.old`

Ahora si podemos cambiar el fichero ejecutando el siguiente comando, donde nos abrirá el editor de texto llamado "nano":

- `sudo nano –w /etc/network/interfaces`

Y ahí, modificamos el contenido dejándolo de tal manera:

```
auto eth0

  iface lo inet loopback

  iface eth0 inet static
  address 192.168.1.75
  netmask 255.255.255.0
  gateway 192.168.1.1
```

Tras ello, guardamos el archivo con los cambios con **Ctrl+o**, salimos del editor con **Ctrl+x** y reiniciamos nuevamente la Raspberry Pi con siguiente comando para que se efectúen los cambios:

- `sudo reboot`

3. Instalación del servidor de ficheros SAMBA

La instalación del servidor de ficheros SAMBA [24] es necesaria para poder compartir fácilmente archivos entra Raspberry Pi y Windows.

El primer paso será instalar el paquete SAMBA:

- `sudo apt-get install samba samba-common-bin`

Conforme se descarga el software e instala, seguidamente empieza a configurarlo automáticamente.

Ahora solo nos falta añadir el usuario "pi" a los usuarios de SAMBA e introducimos la contraseña de red dos veces y el usuario queda añadido:

- `sudo smbpasswd -a pi`

Finalmente, reiniciamos el servidor de ficheros para que tengan efectos los cambios:

- `sudo /etc/init.d/samba restart`

4. Instalación del servicio No-IP

Dado que se va a usar la Raspberry Pi como servidor y en este caso se quiere acceder remotamente desde cualquier red, es más cómodo y seguro crear un subdominio y así no es necesario acceder al dispositivo usando su IP. Además, es posible que la IP del router sea dinámica, es decir, puede variar cada vez que éste se reinicie, por tanto, con este servicio de hostname virtual se puede solucionar el problema de cambio de IP pública.

En primer lugar, es necesario crearse una cuenta gratuita a través de su propia web http://www.noip.com/, y configurar un hostname. Una vez realizado este paso, ya tenemos un usuario y contraseña, por lo que pasamos a instalar el paquete No-IP en la Raspberry Pi ejecutando los siguientes comandos:

- `wget http://www.no-ip.com/client/linux/noip-duc-linux.tar.gz`
- `tar -zxvf noip-duc-linux.tar.gz`

A continuación, accedemos a la carpeta que se ha descomprimido e instalamos el programa:

- `cd noip-2.1.9-1/`
- `make`
- `sudo make install`

Durante el proceso de instalación se pedirá que se introduzca los datos de la cuenta creada anteriormente.

El siguiente paso consiste en crear un fichero con el contenido de la siguiente figura:

- `sudo nano /etc/init.d/noip2`

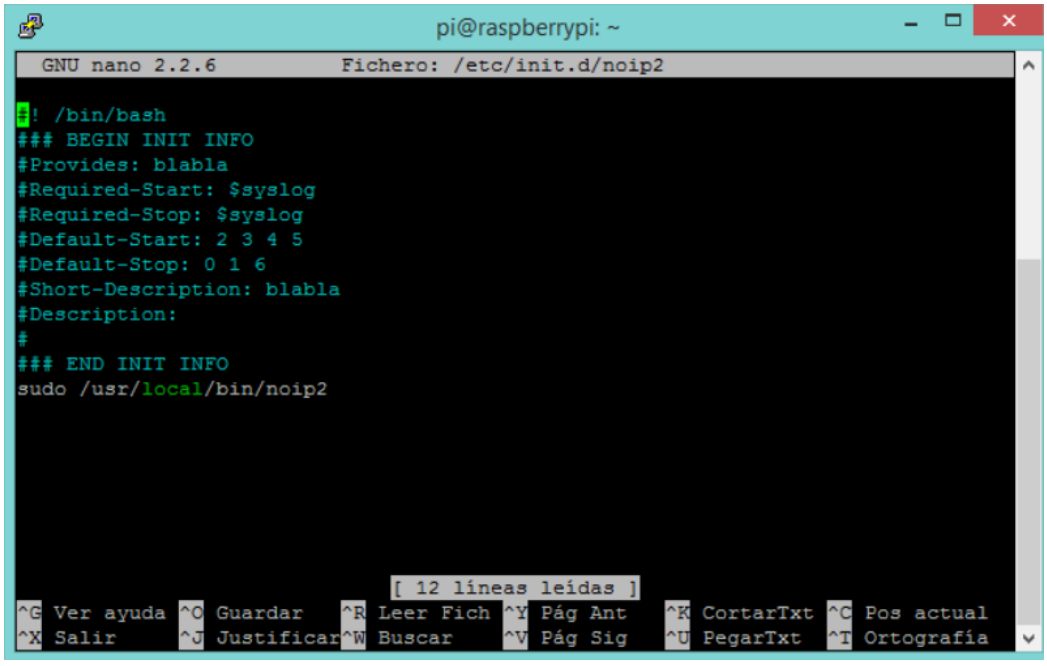

Figura 35 – Fichero de inicio en el arranque del servicio No-IP

Después se guarda el fichero, se le da permiso de ejecución y se coloca en la cola de arranque con las siguientes sentencias:

- `sudo chmod +x /etc/init.d/noip2`
- `sudo update-rc.d noip2 defaults`

Para finalizar se necesita abrir el puerto 80 del router, ya que se usa el protocolo http, al que esté conectada la Raspberry Pi para así poder utilizar el servicio NO-IP.

5. Instalación y configuración de mjpg-streamer

El proceso de instalación y configuración del paquete mjpg-streamer para poder transmitir vía streaming en tiempo real es muy sencillo siempre y cuando se sigan en orden los siguientes pasos:

1. Conectamos la webcam a uno de los puertos USB de la Raspberry Pi.
2. Accedemos a ésta por SSH y nos logamos.
3. Una vez en el terminal, ejecutamos el comando `lsusb` y anotamos el ID de la webcam. En este caso corresponde a la fila cuyo fabricante es Logitech.

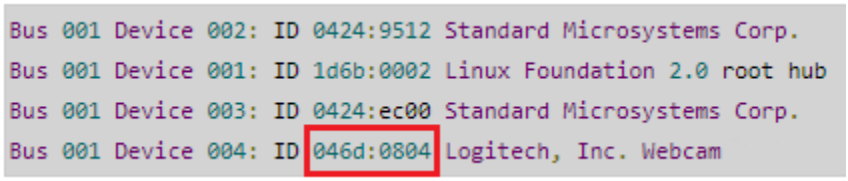

Figura 36 – Salida comando lsusb en Raspberry Pi

4. A continuación, ejecutamos los siguientes comandos para proceder con la instalación:
   - `sudo apt-get update`
   - `sudo apt-get install libjpeg8-dev imagemagick subversion`
   - `mkdir mjpg`
   - `cd mjpg`
   - `svn co https://svn.code.sf.net/p/mjpg-streamer/code/mjpg-streamer/ mjpg-streamer`
   - `cd mjpg-streamer`
   - `make`
5. Luego, iniciamos la retransmisión ejecutando el siguiente comando:
   - `./mjpg_streamer -i "./input_uvc.so -d /dev/video0 -r 320x232" -o "./output_http.so -w ./www -p 8081"`
6. Finalmente, solo falta acceder desde cualquier navegador web a la dirección http://ip-raspberry:8080, sustituyendo ip-raspberry por la IP de nuestra Raspberry Pi. Comentar que esto no es necesario ya que se ha creado una web propia en la que se ve la retransmisión con solo iniciarla.

6. Hacer copia de seguridad de la tarjeta SD de Raspberry Pi

Es importante realizar copias de seguridad periódicas de la tarjeta SD de Raspberry Pi para evitar pérdidas de archivos, trabajo, tiempo, etc. Para ello, vamos a utilizar también la herramienta Win32 Disk Manager.

Para esta utilidad, el proceso consiste en escoger la ubicación de la memoria en "Device" (1), que normalmente es automático; seleccionamos la ubicación del archivo que se va a generar, darle nombre y ponerle la extensión .img (2); y pulsar "Read" (3).

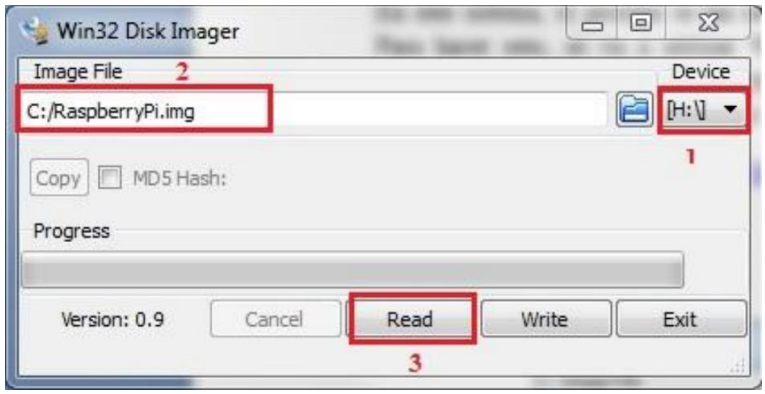

Figura 37 – Pantalla de ejemplo de copia de seguridad de una imagen de la tarjeta SD

Anexo III: Pinouts

- **Pinout Raspberry Pi 3**

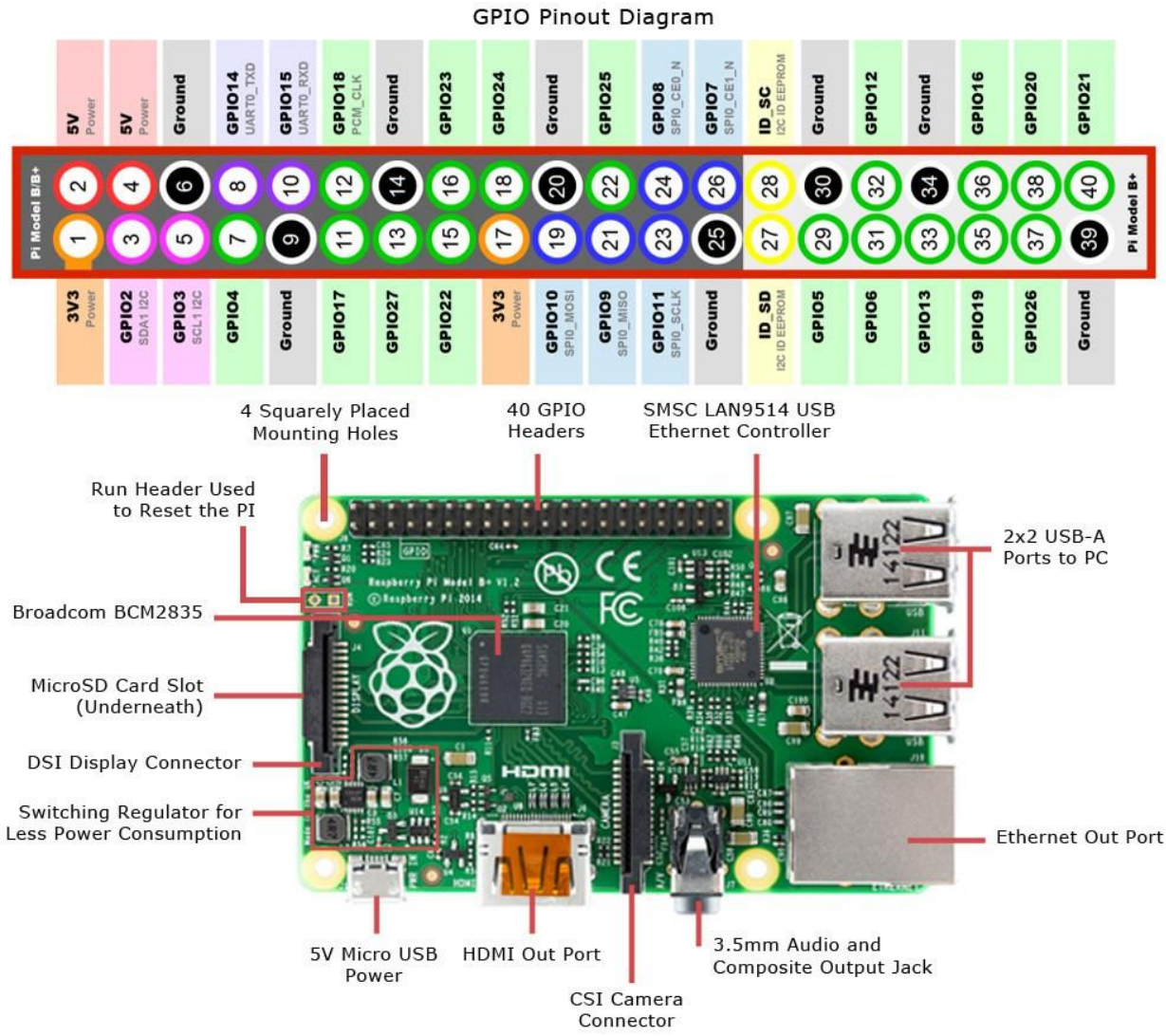

Figura 38 – Pinout Raspberry Pi 3

- **Pinout Arduino UNO**

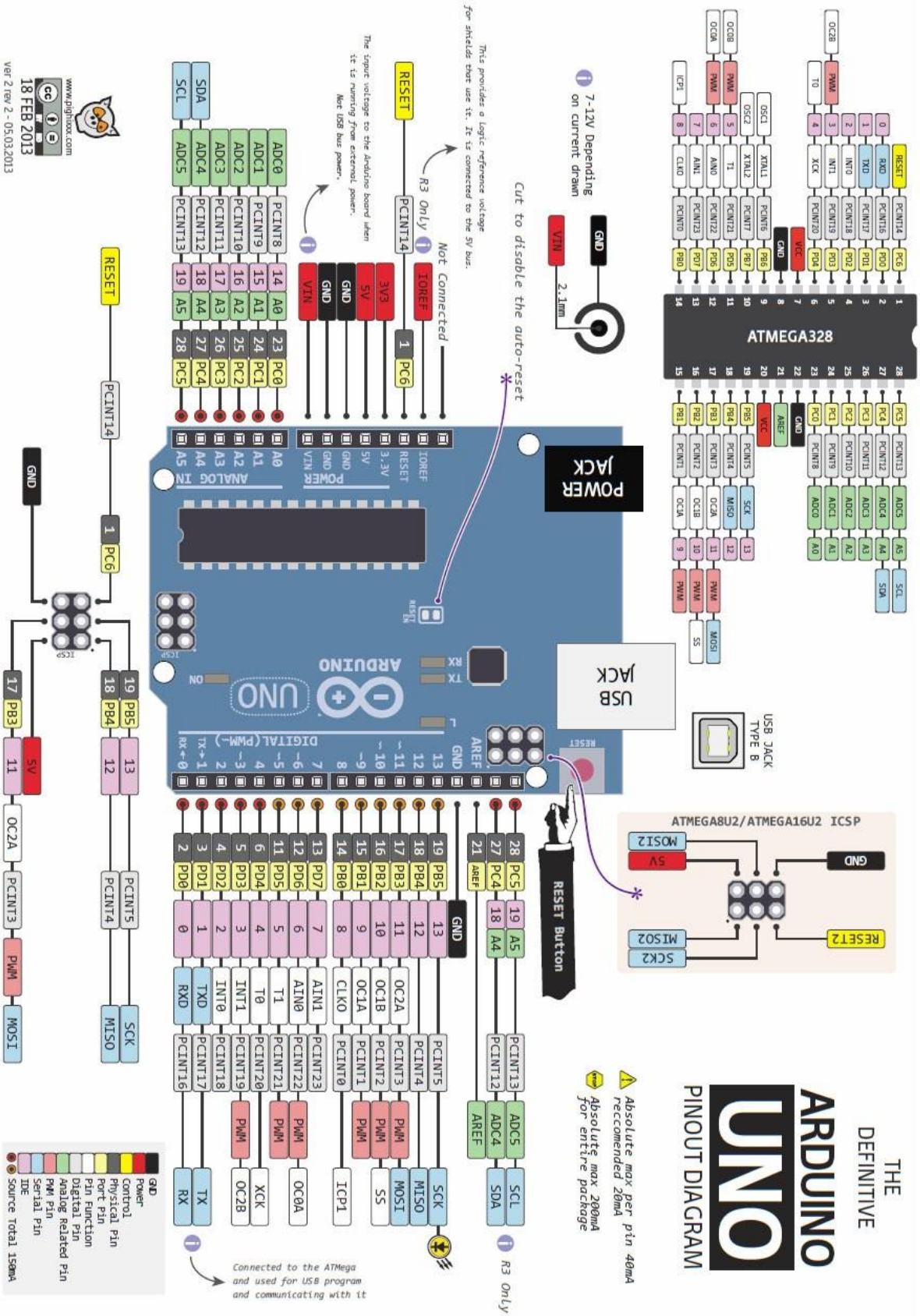

Figura 39 – Pinout Arduino UNO

- **Pinout Arduino YÚN**

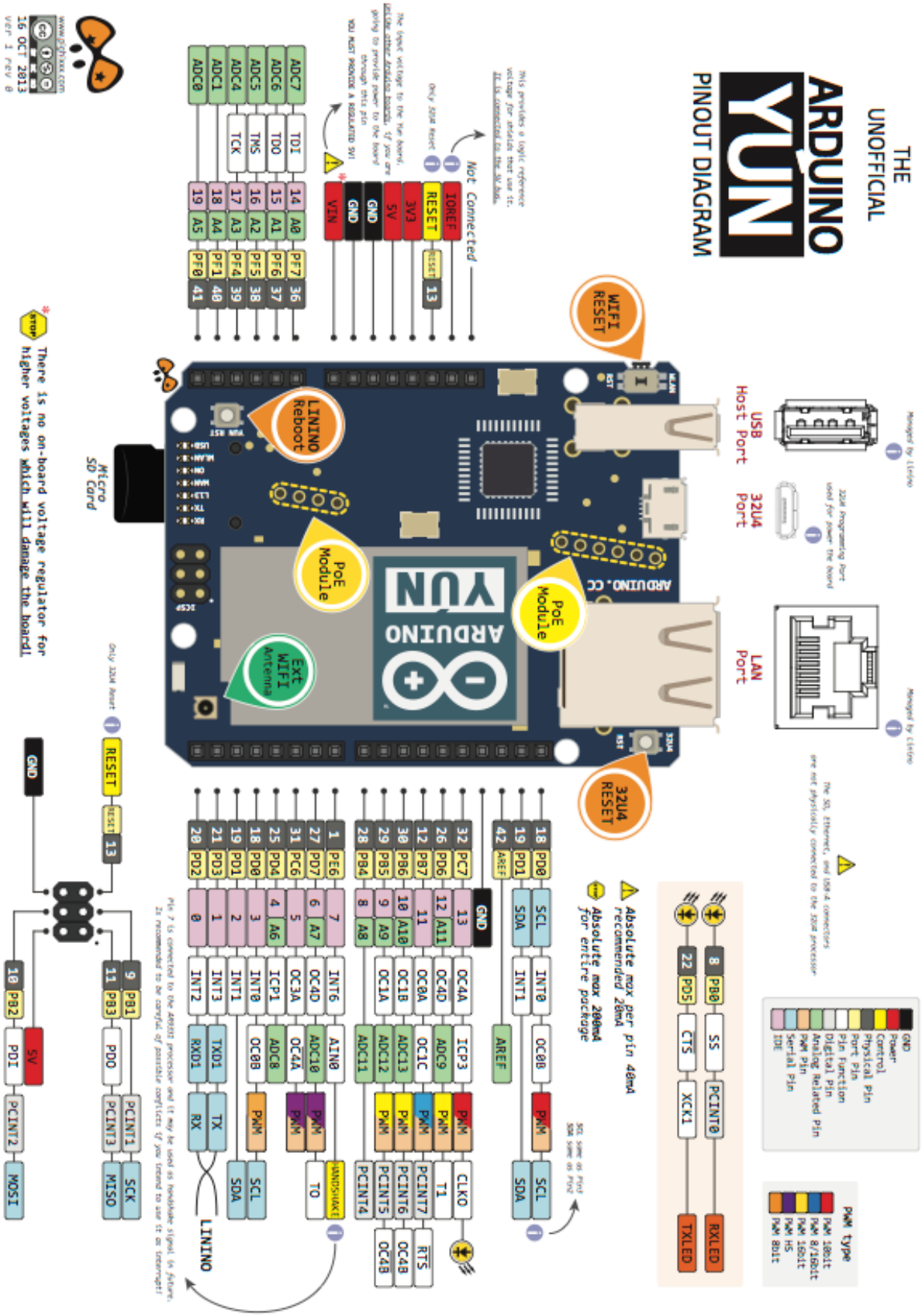

Figura 40 – Pinout Arduino YÚN