UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática
Universitat Politècnica de València

Desarrollo de una red inalámbrica de bajo coste para pequeñas aplicaciones empotradas

Proyecto Final de Carrera

Ingeniería Técnica en Informática de Sistemas

Autor: Alejandro Tomás Colombini Gómez

Director: José Vicente Busquets Mataix

Valencia, Septiembre de 2015

Resumen

El fin de este proyecto era el diseño de una red inalámbrica de nodos. Para ello se empleó una topología de estrella en la que un nodo central controlaba las comunicaciones entre distintos nodos remotos de la red. Estos nodos a su vez disponen de dispositivos periféricos que les permiten interactuar con el mundo externo a la red.

Dado que uno de los objetivos principales era conseguir que la solución desarrollada fuese de bajo coste, se emplearon componentes fácilmente accesibles por su gran popularidad, como lo son las placas Arduino, empleadas para la implementación de los nodos remotos, y la placa Raspberry Pi para el nodo central. Para la comunicación inalámbrica se empleó un módulo basado en el chip NRF24L01+.

Además, se implementó una pequeña aplicación de prueba consistente en dos nodos remotos, uno conectado a un sensor PIR y el otro a una serie de relés, de forma que el estímulo del sensor del primero provoca, por medio de la funcionalidad presentada por la red desarrollada, un cambio en el estado de los relés del otro.

Palabras clave: Red Inalámbrica, Arduino, Raspberry Pi, NRF24L01+, Internet de las Cosas, IoT, Linux Empotrado

Tabla de contenidos

1. INTRODUCCIÓN

El objetivo de este proyecto es desarrollar e implementar una red de nodos de bajo coste. Para ello se buscó la solución más adecuada teniendo en cuenta tanto el precio de los componentes como la complejidad de la implementación.

En primer lugar se decidió cual sería la topología de la red a desarrollar, ya que este parámetro podría hacer variar los componentes empleados y la forma en la que se comunicarían los distintos nodos de la red entre si.

Por tanto, el paso siguiente a la decisión de la topología fue diseñar a grandes rasgos la forma en la que se comunicarían los distintos nodos, para pasar a continuación a la elección de los componentes que se emplearían, tanto en cuestión de hardware como de software.

Para el desarrollo de la parte más relevante y variable del sistema implementado, es decir, los nodos, se tomó como base Arduino, una placa de desarrollo que a un precio muy bajo aúna la potencia suficiente para este proyecto y proporciona una gran facilidad de trabajo, acelerando así tanto el desarrollo de este sistema como su posible futura expansión con la adición de nuevos nodos.

El nodo que se tomó como centro de la red está basado en Raspberry Pi, un SoC con todas las capacidades principales de un PC completo, incluyendo el sistema operativo y la conectividad a través de Ethernet del sistema a Internet. Estas dos características permiten acceder al centro del sistema y alterar su configuración desde cualquier lugar.

La comunicación inalámbrica se realizó por medio de una solución compatible tanto con Arduino como con Raspberry Pi y de fácil uso y conexionado, programable por medio de librerías disponibles para ambos sistemas.

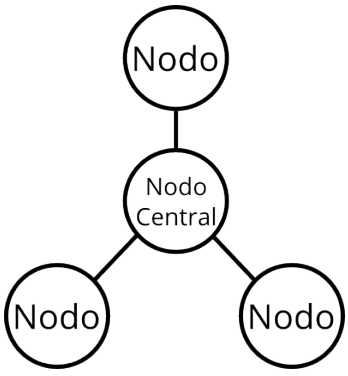
Por último, para probar el sistema implementado. se desarrolló una pequeña aplicación que consistía en la activación y desactivación de relés en un nodo, haciendo uso de la información recogida por otro nodo con un sensor de movimiento.

2. ANÁLISIS Y DISEÑO

El primer paso en el desarrollo del proyecto fue llevar a cabo un análisis del sistema que queríamos construir, así como de las partes que lo componen y las funciones que estas desempeñan. Para ello se tuvo en cuenta que el sistema está compuesto principalmente por un número de nodos independientes que llevan a cabo los distintos procesos de entrada y/o salida en el mundo exterior al sistema y por un medio para comunicarlos inalámbricamente. La combinación de estos dos elementos principales resulta en una red de nodos.

En función de la topología de la red, se puede plantear el sistema de varias formas. Para el desarrollo de este proyecto se decidió emplear una topología de red en estrella.

En este tipo de red hay un nodo central que controla la comunicación entre el resto de nodos. Una vez decidido esto se planteó la forma en que se distribuirían las distintas partes de la funcionalidad de nuestro sistema.



Topología de estrella

Se optó por que los nodos implementasen una funcionalidad propia y se coordinasen por medio de un nodo central. Esto permite una gran modularidad en el sistema, teniendo todos los nodos remotos una interfaz igual con el nodo central y otorgándole a este la flexibilidad suficiente como para no requerir modificaciones en el caso de que se añadan nuevas funcionalidades en los nodos remotos y a la vez aislando cada nodo del resto, lo cual facilita la implementación de cada uno de estos nodos independientes así como la adición de nuevos nodos al sistema.

Una vez decidido esto, consideramos que pudiendo darse la posibilidad de que el nodo central se conectara a Internet, se podría acceder a la configuración del

sistema por medio de la red.

2.1. Diseño de la red

Para llevar a cabo el diseño del sistema descrito se acotaron en primer lugar las partes del sistema que implementarían tanto los nodos remotos como el nodo central. Como se ha adelantado anteriormente, para la topología escogida, es necesaria una interfaz común a todos los nodos para realizar mediante esta la comunicación con el nodo central, que a su vez se encarga de distribuir los mensajes de la red dependiendo de la configuración establecida previamente. Por tanto, se deberá diseñar esta interfaz, que estará presente en cada uno de los nodos remotos, y también la funcionalidad de comunicación entre nodos remotos y la configuración de esta, que formarán parte de la implementación del nodo central.

Dentro de la implementación de los nodos, se pueden diferenciar dos partes concretas.

En primer lugar contamos con la funcionalidad común a todos los nodos, que es la de llevar a cabo la comunicación con el nodo central.

Por otra parte se encuentra la funcionalidad única y específica de cada nodo, esta parte es independiente de la forma en la que se comuniquen el nodo y el resto de la red y depende únicamente de la aplicación para la que se vaya a emplear cada nodo. Esta parte de la implementación de cada nodo interactúa con el mundo exterior al sistema en forma de dispositivos periféricos con una o varias entradas o salidas. Estas entradas y salidas consisten básicamente en entradas o salidas de propósito general o en entradas analógicas. Más adelante, en la sección correspondiente se desarrollará un ejemplo de esto.

Como resultado de estos procesos externos propios de cada nodo se generan o consumen los datos que se comunican entre los nodos, y es por medio de estos que se cohesionan las dos partes principales de cada nodo.

El nodo central del sistema es distinto a los nodos remotos. Este se encarga de distribuir los distintos mensajes que llegan de unos nodos remotos a otros, que atendiendo a una configuración especificada previamente requieran de esta información. Por tanto este nodo implementará la distribución de la información y la configuración de dicha distribución.

Para concretar el funcionamiento de las partes descritas del sistema debemos pasar al diseño de la interfaz entre los nodos remotos y el central, así como la reglas que se definen en la configuración mencionada, según las cuales este último controla la información entre distintos nodos.

2.1.1. Nodos remotos

Para dotar de modularidad al sistema se decidió que los nodos remotos no se conocieran entre sí, esto nos permite la modificación y/o sustitución de nodos sin que se requiera una modificación en todos los nodos dependientes de estos. Esto también permite que los nodos se centren en realizar sus funciones asegurando únicamente la comunicación con el nodo central y por tanto la futura implementación y adición de nuevos módulos será más sencilla.

La comunicación se realiza por medio de mensajes, que se envían para notificar o modificar, respectivamente, el estado de una entrada o de una salida del sistema.

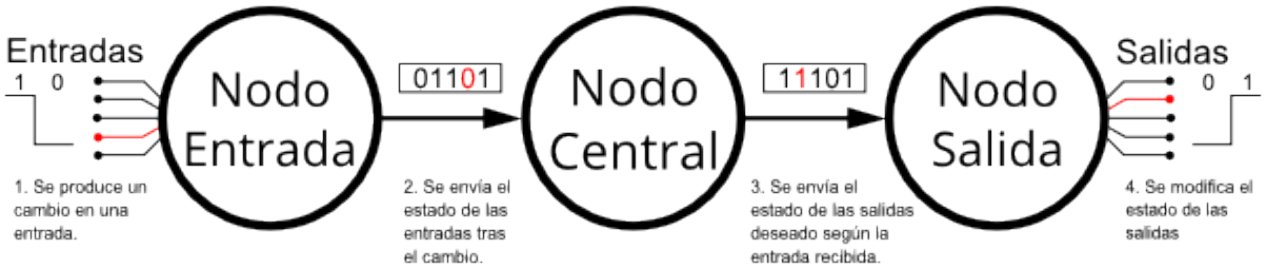
Tal como se ha dicho antes, los nodos pueden contar con una o varias entradas o salidas, esto implica que hay dos tipos distintos de nodos: de entrada, en los cuales se atiende al estado de aquellas de sus entradas relevantes para el sistema; de salida, cuyas salidas se modificarán en función de los valores dispuestos por los nodos de entrada.

Así pues, los mensajes que se intercambian entre los nodos remotos se componen de una serie de campos que representan los estados de estos extremos de la red.

En el caso de los nodos de entrada, cada campo representa el estado que ha tomado una de las entradas de las que dispone el nodo y estos se generan cuando alguno de estos estados cambia.

Para simplificar la configuración del sistema, el valor que representa el estado de cada entrada analógica se debe decidir y aplicar en el nodo remoto .Por tanto en los mensajes este valor está representado, al igual que los estados de las entradas digitales, como activo o inactivo.

En el caso de los nodos de salida, los valores presentados indican el estado al que deben pasar las salidas del nodo.



Esquema del funcionamiento de los nodos remotos

2.1.2. Nodo central

El nodo central de la red cuenta, aparte de con el medio para comunicarse con los nodos remotos del sistema, con una configuración propia.

Esta configuración tiene dos objetivos, en primer lugar, permite identificar los distintos nodos remotos que forman parte del sistema, además de esto, permite vincular las entradas y las salidas del sistema entre si, de forma que ante un cambio en una entrada concreta de un nodo remoto, se altere la salida seleccionada por esta configuración en otro nodo, como se ha descrito en el apartado anterior.

Dados los dos fines de esta configuración se diferencian estas dos partes en dos ficheros distintos. En ambos casos se define una sintaxis sencilla con tal de facilitar la elaboración y modificación del funcionamiento del sistema, así como la adición de nuevos nodos.

El apartado de la configuración dedicado a la identificación de los nodos cuenta con una línea por nodo presente en la red. Cada una de estas líneas consta de dos campos, el primero de ellos indica el tipo de nodo, es decir, si es de entrada o de salida, mientras que el segundo contiene la dirección del nodo.

Para la configuración de los vínculos entre las entradas y las salidas de los respectivos nodos, definimos y nos referiremos a cada uno de estos vínculos como un 'enlace', de forma que una entrada de un nodo se *enlaza* a través de esta configuración a la salida de otro nodo.

Como en el caso de la configuración de los nodos, la configuración de los enlaces entre ellos se indica en una línea por enlace.

La línea de configuración de cada enlace de consta de cuatro campos:

- La dirección de un nodo de entrada.
- El número de entrada, de este nodo, que se quiere enlazar.
- La dirección del nodo de salida al que hace referencia la conexión.
- El número de salida del nodo de salida cuyo estado se modificará como resultado de un cambio en la entrada.

Esta configuración se encuentra almacenada en el nodo central de la red y se carga durante la inicialización del mismo.

Estas son las bases sobre las que se desarrolló el sistema presentado en este proyecto. A continuación se describe brevemente, a modo de caso de pruebas del sistema, una pequeña aplicación para este.

2.2. Aplicación del sistema

Dado que el sistema no cumple más objetivo que el de comunicar entre si los nodos y establecer una interfaz común entre estos y de dar al usuario la posibilidad de configurar esta interacción, para realizar una implementación adecuada se consideró necesario desarrollar como parte del proyecto una aplicación para el sistema, es decir, utilizar el sistema desarrollado como solución para un problema concreto.

La aplicación que se escogió fue una pequeña solución domótica para el control del estado de un grupo de relés. Con esto tenemos un nodo de salida del sistema.

Como dispositivo para el nodo de entrada se escogió un sensor de infrarrojos, de forma que al entrar un objeto en su rango, se modifique el estado uno o varios de los relés del nodo de salida de la aplicación.

Esta configuración nos permitiría implementar, por ejemplo, una solución de iluminación con detección de movimiento, que podemos encontrar en multitud de lugares.

3. HARDWARE Y SOFTWARE EMPLEADOS

Para llevar a cabo la implementación del sistema propuesto, hizo falta elegir un micro-controlador adecuado para llevar a cabo las funciones descritas de los nodos y de alguna solución de comunicación inalámbrica para que los nodos remotos puedan estar en contacto con el central.

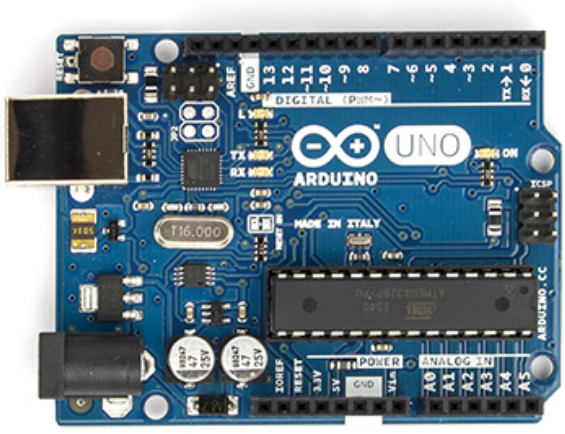
También fue necesario determinar como se implementaría el nodo central, ya que se debe poder configurar de forma sencilla y este requisito aumenta la complejidad de este nodo respecto a los nodos remotos.

Aparte de estos elementos, necesarios para implementar el sistema, se requirió también de los distintos componentes empleados para el desarrollo de la aplicación descrita.

A continuación se describen las herramientas y librerías de software y el hardware empleados en el desarrollo del proyecto.

3.1. Nodos remotos

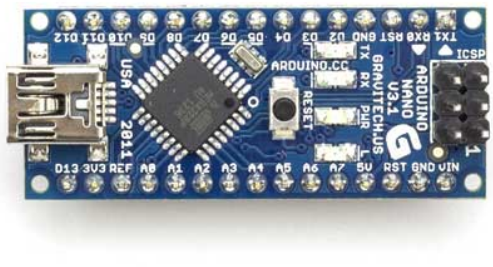
Para la realización del proyecto debíamos elegir una plataforma hardware sobre la cual trabajar. Para ello escogimos la plataforma **Arduino** por las múltiples ventajas que ofrece.



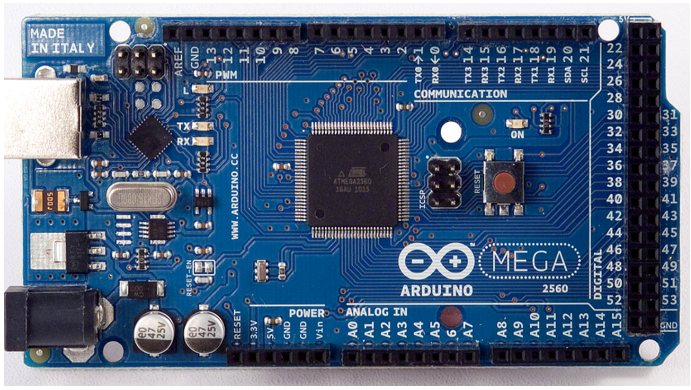
Arduino Uno, el modelo más popular de Arduino

En primer lugar, ofrece una forma sencilla de emplear un micro-controlador con potencia de sobra para el proyecto que deseamos llevar a cabo. El *workflow* que ofrece es rápido, ya que la programación es sencilla y permite un acceso a todas sus funciones con una API simple y con la cual podemos prototipar y probar la programación del controlador con una agilidad que no ofrecen las soluciones más clásicas.

Por otra parte, Arduino dispone de una enorme comunidad de usuarios en el mundo y es fácil encontrar soluciones a problemas frecuentes, desde librerías para todo tipo de uso hasta resolución de dudas, a través de Internet. Una ventaja adicional es contar con distintos formatos de placa, con lo cual se puede adaptar el propio hardware al proyecto a las distintas aplicaciones, con los distintos tamaños disponibles. También hay modelos con mayor funcionalidad para adaptarse a proyectos que lo necesiten.



Arduino Nano



Arduino Mega

También cabe destacar el hecho de que esta es una plataforma de Open Hardware, con lo cual una iteración posterior al alcance de este proyecto podría aprovechar la implementación software realizada en el mismo con pocas o ninguna modificación y a su vez iterar sobre el hardware del proyecto Arduino ya que todos los diseños son accesibles y sus creadores permiten su modificación y posterior distribución.

En nuestro caso se empleó un Arduino Uno y un Arduino Nano para la implementación de los nodos remotos. Ambas placas están basadas en el micro-controlador Atmega328P y cuentan con las mismas características principales:

- 14 entradas/salidas digitales de propósito general, algunas de las cuales tienen funciones adicionales, como puerto serie, PWM o SPI.

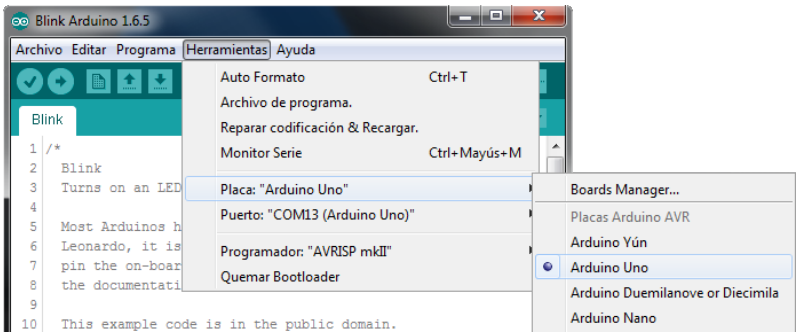
- 8 entradas analógicas.

- Programación del micro-controlador por puerto USB, sin necesidad de un programador externo.

- Alimentación por medio del mismo puerto USB empleado para su programación y opcionalmente mediante pines destinados a ello o, en el caso de Arduino Uno, un conector para un adaptador AC/DC.

La programación de las placas Arduino se llevó a cabo usando la herramienta desarrollada para ello por los mismos creadores, llamada también Arduino. Es un IDE sencillo, basado en Processing IDE, con el cual se pueden realizar las funciones básicas de compilar el código y cargar el programa al chip. También dispone de un monitor serie, muy práctico para hacer pruebas rápidas. El lenguaje de programación es una versión simplificada de C++, aunque se puede programar e importar librerías de C++.

La configuración previa de las placas Arduino consiste en la instalación del driver, que se puede obtener desde la página de los creadores. Y la aplicación mencionada, que se encuentra también en dicha página. La configuración de la aplicación para programar la placa es muy sencilla, simplemente se debe buscar la placa que estamos empleando en una lista que nos ofrece la aplicación y el puerto al que está conectada, que solo se muestra en caso de estar presente la placa.



Selección de la placa en la aplicación, se puede observar también el desplegable para la selección del puerto.

Tras esta configuración solo hace falta compilar el código y cargar el resultado de la compilación a la placa. Esto se puede hacer en dos pasos, presionando el botón “Verificar” ( ) que solo lleva a cabo la compilación del código, o en un solo paso, con el botón “Subir” ( ) que hará el paso de verificación, seguido de la carga

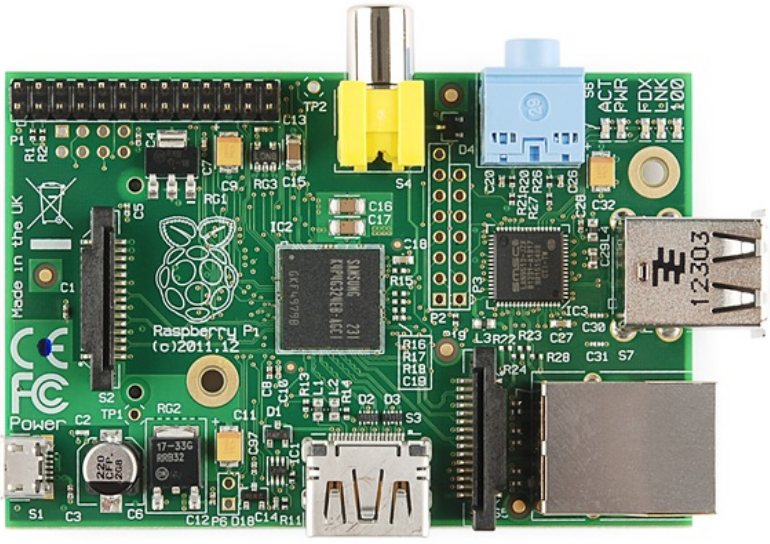
a la placa.

3.2. Nodo central

Como se ha comentado, dadas las características del nodo central del sistema desarrollado, se requería que este contase con capacidad de almacenamiento y que este almacenamiento fuera de fácil acceso y gestión, pues sería ahí donde se pasaría la configuración de la interacción entre los nodos remotos.

Dados estos requisitos, se decidió que la implementación de esta parte del proyecto se realizaría en un **Raspberry Pi**, una placa que integra la funcionalidad de un PC completo en un pequeño dispositivo de bajo coste. Las características que más nos interesan de este dispositivo, aparte del coste, son:

- Una interfaz de entrada y salida de propósito general.
- Puerto Ethernet.
- Uso de sistema operativo, lo que hace más fácil la gestión del almacenamiento y la configuración de las anteriores características.
- Muy bajo consumo de energía.



Raspberry Pi modelo B

Como se ha dicho, la placa Raspberry Pi tiene las características de un pequeño PC, por tanto para su funcionamiento requiere de un sistema operativo compilado para la arquitectura de su procesador, un ARM1176JZFS. Los creadores de Raspberry Pi recomiendan emplear una adaptación hecha por ellos mismos del sistema operativo Debian GNU/Linux. Esta modificación se llama Raspbian y es Debian armel, es decir, Debian compilado para la arquitectura ARM, además de contar con paquetes específicos para esta placa, su configuración y sus usos más habituales.

Raspberry Pi utiliza para el almacenamiento una tarjeta SD, donde se debe instalar el sistema operativo. Para ello los desarrolladores de la placa han creado una aplicación con la cual, disponiendo de la imagen del sistema operativo Raspbian proporcionada en su página web, podemos instalarlo en la SD fácilmente.*

Una vez instalado el sistema operativo se inicia la placa y se configuran los parámetros adecuados, en nuestro caso configuramos Raspbian con el software y los servicios necesarios para hacer uso de sus puertos de entrada y salida de propósito general y para poder acceder a la línea de comandos del sistema operativo por *Ethernet*, mediante *ssh*. También se configuró una dirección IP estática, para tener la seguridad de que el acceso por LAN se realizaría siempre por la misma dirección. Los pasos necesarios para estas configuraciones se detallarán más adelante, en la sección del proyecto dedicada a la implementación.

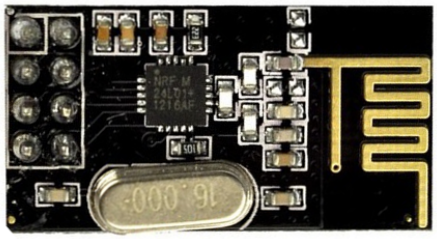
3.3. Comunicación Inalámbrica

Tras elegir la plataforma sobre la cual trabajaríamos, era necesario encontrar un medio para comunicar los distintos nodos. Gracias a la popularidad de la que goza el hardware Arduino actualmente es fácil encontrar una variedad de opciones para la comunicación inalámbrica entre distintas placas Arduino.

Tras valorar las posibilidades que ofrece cada una de las opciones disponibles, nos decantamos por un módulo con el chip NRF24L01+, ya que ofrecía potencia suficiente para cumplir con los requisitos del proyecto y cuenta con una interfaz de programación adaptada a Arduino y bien documentada, todo ello a un coste muy bajo.

En cuanto al hardware del chip NRF24L01+, podemos destacar:

- Transceptor en un chip de 2,4GHz.
- Velocidades de transmisión de 250 kbps, 1Mbps y 2Mbps.
- Muy bajo consumo.



Módulo de comunicación inalámbrica con el chip NRF24L01+.

La misma librería empleada para programar las funciones de red de Arduino cuenta también con una API C++ y Phyton para Raspberry Pi, con lo cual el proceso de integración entre las distintas partes del sistema implementado es casi inmediato.

El conexionado del módulo de comunicación inalámbrica con Arduino y con Raspberry Pi se realizan por medio del bus SPI y se verá más adelante, en el apartado dedicado a la implementación del sistema.

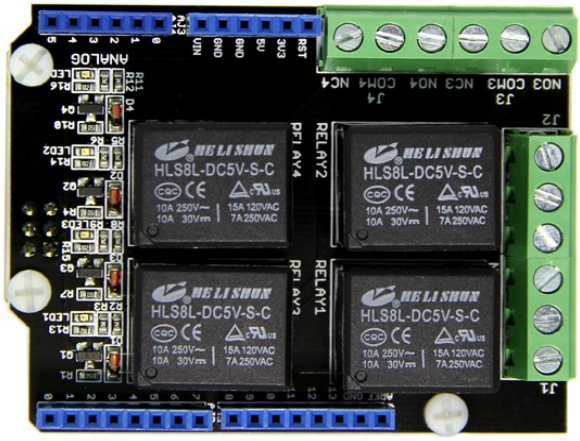
La librería empleada para la programación de estos módulos se llama TMRh20, que a su vez es una mejora sobre otra librería llamada RF24, estas hacen disponibles las funciones que otorga el NRF24L01+ con una API similar a la de las librerías estándar de Arduino.

3.4. Componentes de la aplicación

Como aplicación se implementaron dos nodos, en uno de ellos habría un grupo de relés y en el otro un sensor de movimiento.

Para el nodo de los relés se empleó un componente específico para Arduino que integra cuatro relés en una placa que se sitúa sobre la placa Arduino Uno. Este tipo de componentes está muy extendido en la comunidad de Arduino porque son componentes probados y bien adaptados a la placa y se conocen como *Shields*. En

este caso se empleó el llamado Relay Shield v2.0. Los *Shields* se acoplan encima de la placa, conectándose a todos sus pines y dejando los pines no empleados disponibles para otros usos.



El Relay Shield v2.0 cuenta con cuatro relés y se conecta sobre los pines del Arduino Uno.

Para el otro nodo se recurrió a un sensor de movimiento por infrarrojos, también llamado PIR, del inglés *Passive Infrared Sensor*. Este sensor se conectó a la placa Arduino Nano y se emplearía como entrada del sistema para esta aplicación.

El módulo PIR HC-SR501, empleado como entrada de la aplicación

4. IMPLEMENTACIÓN

4.1. Red de nodos

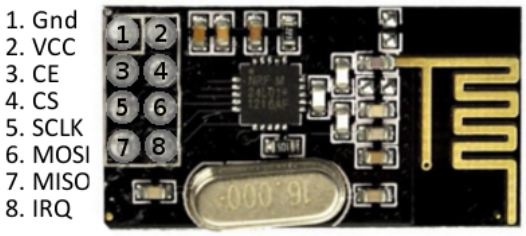
4.1.1. Conexionado de los módulos NRF24L01+

El primer paso para la implementación de nuestra red, fue realizar el conexionado de los módulos NRF24L01+ a las distintas placas que se emplearon. Para realizar un conexionado correcto se consultó la página de los desarrolladores de la librería empleada.

En primer lugar se conectó un módulo a las placas Arduino, cabe tener el cuenta que, al tener la misma disposición de pines, el conexionado es similar entre el Arduino Uno y el Arduino Nano.

Dado que más adelante el nodo de los relés requeriría el Relay Shield, este se conectó desde el principio, ya que el conexionado de los pines propuesto por los desarrolladores de la librería para el módulo de comunicación inalámbrica se debe modificar una vez conectado el *Shield* porque este emplea los pines D4, D5, D6 y D7 de Arduino y a su vez, uno de los pines propuestos es el D7. Para esto se debe llamar a la función de inicialización del controlador de la radio en el código de los nodos indicando los pines que se emplearán, aparte de los dedicados al bus SPI.

El paso siguiente fue realizar la conexión entre el módulo de comunicación inalámbrica y el Raspberry Pi. A continuación se muestran las correspondencias entre los pines de las distintas placas.



Disposición de los pines del módulo de comunicación inalámbrica.

Pines relevantes de la Raspberry Pi

Módulo NRF24L01+	Arduino Uno/Nano	Raspberry Pi
1	Gnd	25
2	3V3	17
3	D9	15
4	D10	24
5	D13	23
6	D11	19
7	D12	21
8	-	-

Conexionado del módulo de comunicación inalámbrica y las distintas placas empleadas como nodos.

Como se ha comentado en el apartado de diseño del sistema, cada nodo remoto contaría con un determinado número de entradas y salidas, que podía poner a disposición del sistema. Teniendo en cuenta los pines requeridos para la conexión del módulo de comunicación inalámbrica, los nodos remotos basados en Arduino cuentan finalmente, como máximo, con nueve entradas/salidas de propósito general (de la D0 a la D8) y con sus ocho entradas analógicas.

Una vez resuelto el apartado de hardware, se pasó a implementar el software que controlaría tanto los nodos remotos como el central.

4.1.2. Librería para la programación de los nodos

Para la programación de la funcionalidad los nodos se decidió crear una pequeña librería con el objetivo de simplificar considerablemente la futura implementación de nuevos nodos. Esta se encargaría a su vez de llamar a las funciones de la librería TMRh20, por medio de la cual se implementan las conexiones entre nodos.

Dado que la librería TMRh20 cuenta con una funcionalidad automática para la confirmación de recepción de mensajes que bloquea el envío hasta que este se confirma o expira un *timeout*, se simplifica bastante la implementación de los envíos de mensajes y su gestión. Además, la librería dispone también de verificación automática de los mensajes mediante CRC, con lo que la comprobación de la integridad de estos se realiza de forma completamente automatizada y opaca al programador de los nodos.

La librería desarrollada presenta la clase principal Node, a la cual instancian todos los nodos de la red durante su inicialización, y define ciertos tipos y funciones para hacer opaco de cara al usuario de la misma el uso de la librería de red. Esto permitirá, además de reducir la complejidad del código de los nodos, que en caso de darse cualquier modificación de su funcionamiento interno, los cambios requeridos en el código implementado para los nodos sean mínimos.

Una de las definiciones realizadas por la librería es la dirección del nodo central, que es conocida por todos los nodos e invariable. Esta dirección no debe emplearse para identificar a ningún otro nodo ya que esto conllevaría un comportamiento indeseado.

La librería define y emplea los siguientes tipos de datos:

- **`Node`**: es la clase principal de la librería de la que cada nodo tiene una instancia y por medio de la cual se controlan todas las comunicaciones entre ellos.

- **`NodeType`**: puede tener valor `CENTRAL_NODE`, `INPUT_NODE` o `OUTPUT_NODE`, representa si el nodo es central, de entrada o salida del sistema. Por supuesto, el valor `CENTRAL_NODE` solo se empleará para el nodo central.

- **`NodeAddress`**: un tipo creado para encapsular las direcciones de 3 bytes empleadas para la identificación de los nodos por parte de la librería. Debido al funcionamiento de la librería, todos los nodos usarán direcciones iguales en todo excepto el primer byte.

- **`NodeMessage`**: es el tipo de dato que se transmite entre los nodos remotos y el central de la red. Consiste en un entero de 32 bits emplado para almacenar el estado (activo/inactivo) de las entradas en los nodos de entrada y de las salidas en los de salida. Los 8 bits de menor peso representan las entradas analógicas, mientras que los 9 siguientes representan las entradas digitales.

- **`NodeFlags`**: flags empleados para aplicar las máscaras de bits necesarias para la interpretación del contenido de los `NodeMessage`.

Los métodos principales con los que cuenta la clase `Node` son:

- **`Node()`**: este es el constructor de la clase `Node` configura el hardware y crea una instancia global del controlador de la radio, que se empleará de manera opaca en el resto de funciones. Aquí se especifica el tipo de nodo y su dirección.

- **`begin`**`()`: según el tipo de nodo especificado en el constructor, este método realizará la inicialización adecuada del mismo. El nombre del método se debe al estándar de las librerías de Arduino.

- **`receive`**`()`: método para recibir datos del nodo central, lo emplean los nodos de salida. Devuelve el mensaje recibido en forma de `NodeMessage`, aplicando las máscaras deseadas usando `NodeFlags` se puede obtener el estado de las salidas requeridas para el nodo.

```
Node node{OUTPUT_NODE, NODE_ADDRESS};

// [...]

NodeMessage msg;
if (msg = node.receive ()) {
// Se aplica el flag NODE_D7 de NodeFlags al mensaje para comprobar
// si ese campo está activo
    if (msg & NODE_D7)
// Si lo está, se establece el estado del pin 7 a alto
       digitalWrite (7, HIGH);
    else
// Si no, se establece el estado del pin 7 a bajo
        digitalWrite (7, LOW);
}
```

- **`send`**`()`: si se produce un cambio en sus entradas, los nodos de entrada emplean este método para informar del cambio al nodo central para que este informe a los nodos remotos que dependan de este nodo.

```
// Se toma el estado de una entrada
int pin_2 = digitalRead (2)
int flags = 0;

// [...]

// Si está a nivel alto, se indica que se ha activando el flag NODE_D2
if (pin_2 == HIGH)
    flags |= NODE_D2;
else if (pin_2 == LOW)
    flags |= ~NODE_D2;

// Se envían los flags con el estado deseado al nodo central
node.send (flags);
```

- **`sleep`**`()`: este método activa el modo de bajo consumo de la radio y detiene el nodo durante el tiempo especificado, en milisegundos. Se emplea para el ahorro de energía de los nodos remotos.

- **`awake`**`()`: recupera el funcionamiento normal de la radio después de haber llamado al método `sleep()`.

```
void loop () {
// Pasa a modo de funcionamiento normal
    awake ();

// [...] Código propio del nodo

// Pasa a modo de bajo consumo de energía durante medio segundo
    sleep (500);
}
```

Así pues, con la funcionalidad presentada en esta librería y haciendo uso de las librerías estándar de acceso al hardware propias de Arduino, se podrán implementar los nodos. Puesto que más adelante, se detalla el desarrollo de la aplicación implementada, en ese apartado se darán más detalles de la implementación de cada tipo de nodo.

4.1.3. Nodo central

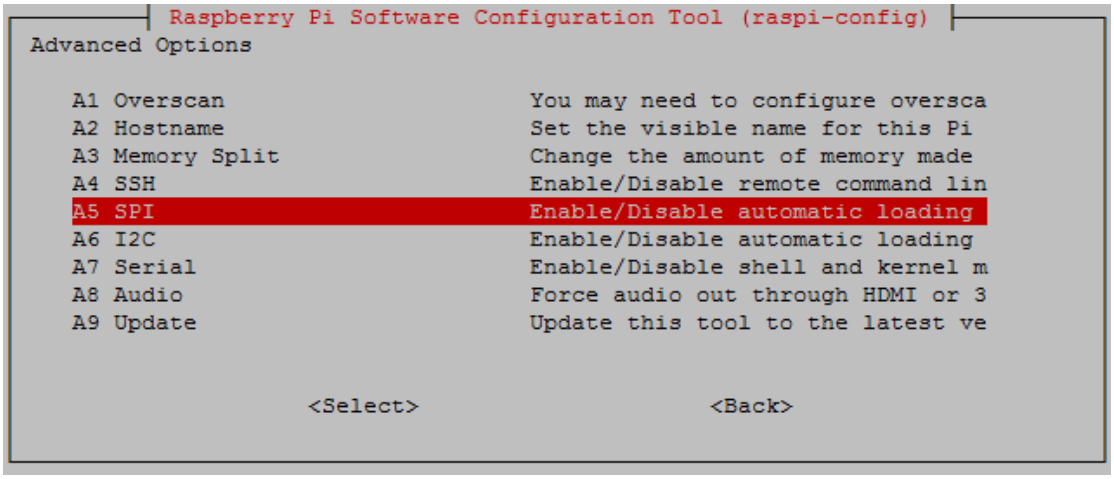
Dado que la implementación que se detallará más adelante de los nodos no

cubre el nodo central por ser este es un caso único, en primer lugar se repasarán los pasos necesarios para su configuración.

Tras instalar el sistema operativo* se inserta la tarjeta SD en la ranura de la placa, se conecta esta a la corriente por medio de un adaptador de 5V y se inicia por primera vez. En este primer arranque se debe conectar la Raspberry Pi a un monitor por medio de su puerto HDMI, ya que la configuración inicial se realiza con una interfaz gráfica. También será necesario conectar a la placa un teclado por USB para controlar la interfaz mencionada. Es recomendable conectar la Raspberry Pi a Internet por Ethernet, para que las configuraciones o actualizaciones que lo requieran se realicen automáticamente.

Una vez se inicia la placa por primera vez se muestra un menú de configuración. Para preparar la Raspberry Pi para trabajar con el módulo de comunicación inalámbrica se debe habilitar la ejecución en el arranque del servicio que controla el SPI. Para ello se debe acceder al apartado “Advanced” de las opciones y dentro del menú que aparece, seleccionar SPI y confirmar.

De la misma manera, en este menú, se debe habilitar la ejecución en el arranque del servicio ssh, que será el medio por el cual nos comunicaremos en adelante con la placa, por medio de Ethernet. Tras ambas operaciones la herramienta de configuración aplicará los cambios necesarios,



Menú de opciones avanzadas de la configuación inicial de Raspberry Pi

El siguiente paso en la configuración de la placa para su posterior uso es el de fijarle una IP estática de modo que siempre podamos acceder a la misma dirección dentro de nuestra red local.

Lo más recomendable para hacer esto es comprobar durante el primer arranque la dirección IP asignada por DHCP a la placa, para ello se empleará el comando *ifconfig*. Una vez conocida la dirección IP dinámica asignada a la Raspberry Pi, accederemos por medio de ssh empleando, si no se ha modificado, el usuario por defecto 'pi', con contraseña 'raspberry'. Para la conexión por medio de ssh se empleó la famosa herramienta libre *PuTTY*.

El proceso de asignación de la IP estática se basa en la modificación del fichero de configuración de interfaces de red de Debian, situado en */etc/network/interfaces*, en el cual se deben modificar las siguientes líneas:

```
auto eth0
allow-hotplug eth0
iface eth0 inet manual
```

Sustituyéndolas por:

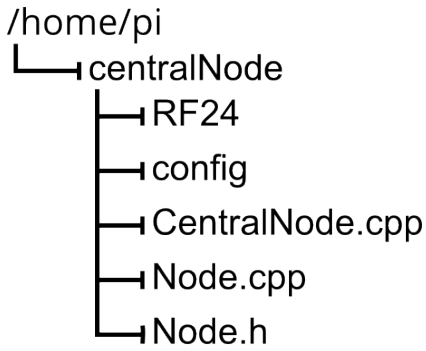
```
auto eth0
allow-hotplug eth0
iface eth0 inet static
address 192.168.1.254
netmask 255.255.255.0
gate 192.168.1.1
```

Con los valores de las dirección, la mascara de enlace adecuados a la red en la que nos encontremos. Tras lo cual reiniciaremos la placa para conectarnos a la nueva IP estática configurada.

Una vez realizados estos pasos, lo restante es realizar la instalación automatizada en la placa de la librería TMRh20, este proceso está explicado en la sección dedicada a Raspberry de la documentación de la librería.*

Tras estas configuraciones, ya podemos pasar a la implementación del nodo central. La estructura del directorio de trabajo que se empleó se muestra a continuación. Se puede observar que el directorio de trabajo principal, *centralNode*, contiene a su vez dos directorios, *RF24*, donde está la librería TMRh20, y *config*, donde se almacenan los ficheros de configuración que se comentarán más adelante. Los archivos *Node.cpp* y *Node.h* son los que contienen la clase `Node`, y

por último el fichero *CentralNode.cpp* es el que contiene el código que rige el funcionamiento del nodo central.



Estuctura del directorio de trabajo en la Raspberry Pi

La librería TMRh20, en sus ejemplos de programación para Raspberry Pi contaba con un fichero *Makefile*, que sirve para automatizar la compilación del software por medio del comando *make*, lo cual ahorra llamar cada vez al compilador (g++) y reescribir las opciones de compilación, además de ofrecer varias opciones adicionales, como la instalación del software y la limpieza de los archivos producidos en la compilación. Este fichero se modificó ligeramente para que compilara nuestro código, para ello se cambiaron dos líneas, en las que se especificaba el nombre de los programas a compilar, que se cambió por *CentralNode* y la dirección donde buscar la librería, que como se ha comentado es el directorio RF24. Este *Makefile* se situará dentro del directorio centralNode.

Como se ha observado y al igual que los nodos remotos, el nodo central emplea la clase `Node` y sus métodos, aunque la implementación interna de estos métodos es distinta si el tipo de nodo especificado en la llamada al constructor es `CENTRAL_NODE`, por ello se describe aquí su comportamiento único.

La principal diferencia entre el funcionamiento de los nodos remotos y del central es que los primeros se comunican únicamente en una dirección (enviando o recibiendo mensajes) con el nodo central, mientras que este se comunica tanto con nodos de entrada como con nodos de salida y por tanto recibe y envía los mensajes.

Tal y como se indicó en el apartado del diseño del nodo central, esto requiere de información adicional. A continuación se recuerda someramente la información necesaria para pasar después a la descripción de la implementación de la

configuración propuesta en la etapa de diseño.

En primer lugar, se requiere de una configuración que ayude a determinar al nodo central cómo se debe comunicar con cada nodo, dependiendo tanto de que tipo de nodo se trata como de su dirección.

Así mismo, para informar al nodo central como se relacionan los nodos entre sí, y que este pueda procesar los mensajes procedentes de los nodos de entrada y generar los de salida, se requiere de una información que vincule las entradas de unos nodos con las salidas de los otros.

Es para almacenar esta información para lo que existe en directorio *config.* En este directorio se almacenan dos ficheros de configuración.

El primero de ellos, llamado *node.conf*, contiene el tipo y la dirección de cada uno de los nodos presentes en la red. Para definir estos datos, se empleó una sintaxis muy simple, que se lee durante la inicialización del nodo. El tipo de nodo se representa con una '`I`' si el nodo es de entrada y con una '`O`' si es de salida. Y la dirección se especifican los 3 bytes en hexadecimal, de forma parecida a como se hace en el código de los nodos. Por tanto, la representación de un nodo de entrada y uno de salida sería similar a la siguiente:

```
- Nodo de entrada
I:07F0F0
- Nodo de salida
O:04F0F0
```

Como se observa en el ejemplo, se ha implementado también una sintaxis para añadir comentarios a las configuraciones, antecediendo a estos un guión (-), mejorando así notablemente la legibilidad de estos ficheros. Aparte, si se encuentra una línea en blanco esta se ignora para permitir que el usuario pueda organizar mejor visualmente las entradas del fichero.

Para la configuración de lo que en el apartado de diseño llamamos enlaces, se creó el fichero *link.conf* para el cual se empleó una sintaxis parecida. Pero dado que en cada entrada de este fichero hay cuatro campos, se emplearon dos separadores distintos, permitiendo así que se diferencie cada uno de los nodos involucrados en el enlace. El fragmento de este fichero podría ser el siguiente:

```
- Enlaza el pin 2 del nodo 01F0F0 al pin 7 del nodo 02F0F0
01F0F0:D2-02F0F0:D7
```

Durante su inicialización, el nodo central carga la información de *node.conf* a un vector de objetos de tipo `Node` y los enlaces se guardan dentro en un vector de estructuras `Link`, que almacenan los dos `Node` y los `NodeFlags` que representan los pines indicados en el archivo, en este caso NODE_D2 y NODE_D7. Así si el cambio en un pin de entrada modifica más de un pin en la salida, se pueden añadir los `NodeFlags` necesarios para su activación sin necesidad de crear otro `Link`.

Tras esta carga inicial, el nodo central entra en modo de escucha de los nodos de entrada. A partir de entonces, si algún nodo le envía un mensaje, se consulta la colección de `Link`s y se envía el mensaje con los flags necesarios al nodo de salida correspondiente.

4.2. Aplicación

A continuación se describirá la implementación de la aplicación que se desarrolló como ejemplo de uso de la red implementada y que por tanto servirá para ejemplificar la implementación de los nodos remotos de la red.

En ambos casos, tanto en nodos de entrada como de salida, los primeros pasos para la implementación son parecidos.

Lo primero que se hará es crear una variable global de tipo `NodeAddress`, para almacenar la dirección del nodo.

```
// Dirección de este nodo
NodeAddress NODE_ADDRESS = {0x02, 0xF0, 0xF0};
```

A continuación, se creará un objeto `Node`, también global, en cuyo constructor se indica si el nodo es de entrada o de salida y la dirección del mismo.

```
// Indica INPUT_NODE o OUTPUT_NODE
Node node (INPUT_NODE, NODE_ADDRESS);
```

El siguiente punto en común entre ambos tipos de nodo se da en el método `setup()` de Arduino. Aquí, en ambos casos, se llama al método `begin ()`. A partir de este punto cada nodo realizará las configuraciones y acciones necesarias para su funcionamiento propio.

En el nodo de entrada se ha conectado al pin D2 de la placa Arduino el sensor de infrarrojos, por tanto, cuando se detecte que este pin está a nivel alto se informará al nodo central con un mensaje. Para ello se debe indicar en `setup()` que este pin funcionará como entrada.

A continuación dentro de la función `loop()`, que es la correspondiente al bucle principal del programa del microcontrolador, se engloba el código correspondiente a la lectura del sensor dentro de las llamadas `awake()` y `sleep()`, indicando en el último un paso a modo de bajo consumo durante un segundo. Esto hará que el nodo solo lea el sensor PIR cada segundo y que se mantenga inactivo entre lecturas, permitiendo un menor consumo de energía. Un ejemplo de esto se vio junto a la descripción de los métodos de la clase `Node`.

Cuando el nodo vuelva a su funcionamiento normal, se leerá la entrada D2, correspondiente al sensor y se pasará su estado, por medio de un mensaje con el `NodeFlag` `NODE_D2` y empleando el método `send()`, al nodo central.

Para corresponder a esta entrada, en el nodo central indicaremos, en el fichero *node.conf*, las siguientes líneas:

```
- Nodo PIR
I:01F0F0
- Nodo relés
O:02F0F0
```

Así definimos que la dirección del nodo de entrada, que como se aclara en el comentario previo a su definición, es el conectado el sensor de infrarrojos, mientras que el nodo de salida es el conectado a los relés.

Y para definir los enlaces entre los pines de ambos nodos, escribimos las siguientes lineas en el fichero *link.conf :*

```
- El pin 2 del nodo de entrada activa los pines 7 y 6 del nodo de salida
01F0F0:D2-02F0F0:D7
01F0F0:D2-02F0F0:D6
```

Con esta configuración, cuando el nodo central reciba un mensaje del nodo de entrada con el `NodeFlag` `NODE_D2` activo, enviará un mensaje al nodo de salida con los `NodeFlags` `NODE_D7` y `NODE_D6` activos.

En cuanto al nodo de salida, dado que su conexión se realiza al insertar el *Relay Shield* en el Arduino, los pines conectados al mismo serán siempre el D4, D5, D6 y D7. Por tanto, aparte de la llamada a `node.begin()` común a ambos nodos, el nodo de los relés configurará estos pines como salidas.

En la implementación del nodo de salida este comprueba regularmente si ha llegado algún mensaje desde el nodo central empleando el método `node.receive()`. Cuando este devuelve un mensaje, comprueba sus `NodeFlags`. Si `NODE_D7` está activo, el nodo pone la salida correspondiente al pin D7 a nivel alto y viceversa. Si `NODE_D6` está activo, pone la salida correspondiente a nivel bajo y viceversa.

Con la prueba de esta aplicación se comprobó que la red funcionaba y que otorgaba flexibilidad en sus configuraciones. Ya que si se desea modificar el comportamiento de uno de los nodos se puede modificar uno de los nodos, la configuración del nodo central, ambos nodos, o incluso todo ello.

6. CONCLUSIONES

Con este proyecto se ha conseguido desarrollar una sencilla red de nodos de bajo coste y con cierta flexibilidad, a la vez que fácil de expandir. Se han probado tecnologías y componentes populares en la activa comunidad de Internet y se ha comprobado que realmente son herramientas accesibles y versátiles, motivo por el cual gozan de esta popularidad.

Adicionalmente, cabe tener en cuenta que este proyecto se ha realizado íntegramente empleando software libre y que los componentes hardware empleados, a excepción de algunos chips, son productos creados por colectivos pertenecientes al movimiento Open Hardware, lo cual era un objetivo personal.

De esto se puede concluir que el mundo del hardware y el desarrollo de dispositivos goza de un momento de auge gracias a las comunidades desinteresadas de profesionales que disfrutan apasionadamente de entregar sus conocimientos y esfuerzos al resto del mundo, acercando la tecnología y su desarrollo tanto a nivel económico como por facilidad de desarrollo.

7. BIBLIOGRAFÍA

Arduino. *Página de referencia de Arduino*.
<http://www.arduino.cc/en/Reference/HomePage>

Raspberry Pi Foundation. *Installing Operating System Images*.
<http://www.raspberrypi.org/documentation/installation/installing-images>

Comunidad de Debian, Debian Wiki. *Network Configuration*.
<http://wiki.debian.org/NetworkConfiguration>

TMRh20 Project. *Optimized High Speed NRF24L01+ Driver Class Documenation.*
<http://http://tmrh20.github.io/RF24/>

Seeed Studio. *Relay Shield V2.0.*
<http://www.seeedstudio.com/wiki/Relay_Shield_V2.0>

Nordic Semiconductor. *NRF24L01+ Product Specification.*
<https://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01P>

8. Información

Título del proyecto

Desarrollo de una red inalámbrica de bajo coste para pequeñas aplicaciones empotradas

Alumno

Alejandro Tomás Colombini Gómez

Director

José Vicente Busquets Mataix

Universidad

Universidad Politécnica de Valencia (UPV), campus de Vera

Escuela

Escuel Técnica Superior de Ingeniería Informática (ETSInf)

Departamento

Departamento de Informática de Sistemas y Computadores (DISCA)

Curso

2014 – 2015

Titulación

Ingeniería Técnica en Informática de Sistemas (ITIS)

Intensificación

Informática Industrial