



UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y
DISEÑO INDUSTRIAL

Grado en Ingeniería Electrónica y Automática Industrial

TRABAJO FIN DE GRADO

INTEGRACIÓN DE UN BRAZO ROBÓTICO EN UNA RED DOMÓTICA

José Luis Grande Morón

Cotutor: Miguel Hernando
Gutiérrez
Departamento: Ingeniería
Eléctrica, Electrónica,
Automática y Física Aplicada

Tutor: Alberto Brunete González
Departamento: Ingeniería
Eléctrica, Electrónica,
Automática y Física Aplicada

Madrid, junio, 2019



escuela técnica superior de
ingeniería
y diseño
industrial

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y
DISEÑO INDUSTRIAL

Grado en Ingeniería Electrónica y Automática Industrial

TRABAJO FIN DE GRADO

**INTEGRACIÓN DE UN BRAZO
ROBÓTICO EN UNA RED DOMÓTICA**

Firma Autor

Firma Cotutor (si lo hay)

Firma Tutor

Copyright ©2019. José Luis Grande Morón.

Esta obra está licenciada bajo la licencia Creative Commons

Atribución-NoComercial-SinDerivadas 3.0 Unported (CC BY-NC-ND 3.0). Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-nc-nd/3.0/deed.es> o envíe una carta a Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, EE.UU. Todas las opiniones aquí expresadas son del autor, y no reflejan necesariamente las opiniones de la Universidad Politécnica de Madrid.

Titulo: Integración de un brazo robótico en una red domótica

Autor: José Luis Grande Morón

Tutor: Alberto Brunete González

Cotutor: Miguel Hernando Gutiérrez

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Trabajo Fin de Grado el día de de ... en, en la Escuela Técnica Superior de Ingeniería y Diseño Industrial de la Universidad Politécnica de Madrid, acuerda otorgarle la CALIFICACIÓN de:

VOCAL

SECRETARIO

PRESIDENTE

Agradecimientos

A mis abuelos. María, Sofía, Mariano y José. De todos ellos tengo algo que me ha ayudado durante estos años y que, no tengo duda, seguirá ayudándome. El valor del trabajo, el esfuerzo y el conocimiento.

A mis padres y a mi hermana.

A mis compañeros y sus consecuencias.

A Irene.

A todos ellos, muchas gracias por todo.

Resumen

Este proyecto se resume en.....

Palabras clave: palabraclave1, palabraclave2, palabraclave3.

Índice general

Agradecimientos	IX
Resumen	xI
Índice	xv
1. Introducción	1
1.1. Motivación del proyecto	1
1.2. Objetivos	2
1.3. Materiales utilizados	4
1.3.1. Componentes hardware	4
1.3.2. Componentes software	6
1.4. Estructura del documento	6
2. Marco Teórico	7
2.1. Conceptos de la comunicación serial	7
2.1.1. Características	7
2.1.2. Problemas	10
2.1.3. Usos y aplicaciones	10
2.2. Conceptos de la comunicación por radiofrecuencia	11
2.2.1. Características	11
2.2.2. Problemas	16
2.2.3. Usos y aplicaciones	16
3. Estado del arte	19
3.1. Water level control using Raspberry Pi + XBee + XBMQ + MQTT + Node-Red [9]	19
3.2. Servidor Raspberry Pi-XBee en Python [3]	20
3.3. Smart Porch Light Project [12]	21
3.4. Home Automation System [5]	22
3.5. Controlador central para un sistema domótico utilizando el protocolo inalámbrico ZigBee [2]	23
3.6. Node-RED based custom full-room wake-up light [6]	23
3.7. ArduSmartHome [1]	24
3.8. University of Minnesota – Solar Vehicle [13]	25
4. Desarrollo del proyecto	27
4.1. Planteamiento inicial	27
4.2. Unidad de mando	28

4.2.1. Rasberry Pi 3 Model B	29
4.2.2. MQTT	32
4.2.3. Node-RED	33
4.2.4. Python scripts	44
4.3. Transmisión de la información	45
4.3.1. XBee Shield	45
4.3.2. Módulos XBee	45
4.4. RoboHealth Arm	45
4.4.1. Protocolo de comunicación RHA	45
4.4.2. Modificaciones a RHA	45
4.5. Integración	45
4.5.1. MQTT - Node-RED	45
4.5.2. User Interface - Node-RED	45
4.5.3. Node-Red - XBee	45
4.5.4. XBee - RHA	45
5. Resultados y discusión	47
5.1. Resultados	47
5.1.1. Test User Interface	47
5.1.2. Test MQTT	47
5.2. Discusión	47
6. Gestión del proyecto	49
6.1. Ciclo de vida	49
6.2. Planificación	49
6.2.1. Planificación inicial	49
6.2.2. Planificación final	49
6.3. Presupuesto	49
6.3.1. Personal	49
6.3.2. Material	49
6.3.3. Resumen de costes	49
7. Conclusiones	51
7.1. Conclusión	51
7.2. Desarrollos futuros	51
A. Anexo A	53
A.1. Instalación de Raspbian OS	53
A.2. Instalación MQTT en Raspbian OS	54
A.3. Edición y compilación de scripts en Raspbian OS	55
B. Anexo B	57
B.1. Código SendAT.py	57
B.2. Código SendAPI.py	58
B.3. Código Receive.py	58
C. Anexo C	61
C.1. Datasheet XBee Shield	61

Índice de figuras

1.1. Estructura RoboHealth	2
1.2. Raspberry Pi 3 model B	4
1.3. Arduino Uno R3	4
1.4. XBee Shield	5
1.5. XBee Module	5
1.6. Arduino Mega	5
1.7. Interfaz de edición de Node-RED	6
2.1. Formato serie marca/espacio	8
2.2. Transmisión serial síncrona	8
2.3. Transmisión serial asíncrona	9
2.4. Ejemplo de radiotransmisor AM	12
2.5. Modulación de la señal	13
2.6. Modulación de fase	14
2.7. Banda estrecha vs DSSS	15
2.8. Radiocomunicación simplex a una frecuencia	15
2.9. Radiocomunicación simplex a dos frecuencias	15
2.10. Radiocomunicación semiduplex	15
2.11. Radiocomunicación duplex	16
3.1. Montaje del proyecto 3.1	20
3.2. XBee Shield v2.0 - Seeed Studio	21
3.3. Stack Hardware del proyecto 3.4	22
3.4. Esquema de conexión del sensor de temperatura	22
3.5. PandaBoard ES	23
3.6. Flujo de Node-RED	24
3.7. Arduino Yun	25
3.8. EOS II Solar car	26
4.1. Diagrama de interacción	27
4.2. Diagrama de casos de uso	28
4.3. Diagrama de Secuencia de envío	29
4.4. Layout de puertos de la Raspberry Pi	30
4.5. Escritorio de Raspbian	31
4.6. Interfaz de Putty	32
4.7. Arquitectura MQTT	32
4.8. Ejemplo de interfaz de Node-RED	34
4.9. Nodo Debug	34
4.10. Nodo Function	35

4.11. Nodo Debug	35
4.12. Nodo entrada MQTT	36
4.13. Nodo Inject	36
4.14. Nodo Delay	37
4.15. Nodo Switch	37
4.16. Nodos de entrada	38
4.17. Nodos de salida	38
4.18. UI de control elemental	39
4.19. Flujo de control elemental	40
4.20. Dashboard	40
4.24. Flujo de encendido de RHA	40
4.21. Flujo de control de la cama	41
4.22. Flujo de control de las persianas	41
4.25. Flujo de recepción de RHA	41
4.23. Flujo de control de Zolertia	42
4.26. Flujo de configuración de RHA	43
4.27. Flujo de emisión de RHA	43
4.28. Flujo de control de Zolertia	46
A.1. Descarga de Raspbian Stretch	54
A.2. Versión de Raspbian	54

Índice de tablas

Capítulo 1

Introducción

El presente documento corresponde a la realización de un Trabajo Final del Grado en Electrónica Industrial y Automática basado en la conexión e integración de un brazo robótico en una red domótica. A continuación, se recoge de manera ordenada y detallada el desarrollo e implementación del proyecto; así como los resultados obtenidos y las conclusiones a las que es posible llegar.

1.1. Motivación del proyecto

El punto de partida es el proyecto Robohealth. Consiste en un conjunto de entidades en colaboración para el desarrollo de soluciones relacionadas con la robótica y la domótica con el fin de introducir mejoras en el sistema sanitario. Como se puede observar, entre estas entidades está, además de otras dos universidades públicas de la Comunidad de Madrid, la Universidad Politécnica de Madrid.

Los resultados del proyecto están orientados a pacientes con enfermedades crónicas o capacidades cognitivas limitadas, pacientes en una situación de dependencia a los que es posible mejorar la calidad de vida. Estas mejoras se obtienen a través del diseño y fabricación de robots de asistencia, tanto para pacientes como para sus cuidadores, y la implementación de entornos inteligentes.

En la figura 1.1, se pueden observar los diferentes paquetes de trabajo y subproyectos en los que se trabaja dentro de la estructura de RoboHealth, repartidos entre las entidades colaboradoras. En la Universidad Politécnica de Madrid, encargada del desarrollo de entornos inteligentes de asistencia y rehabilitación, se ha venido trabajando en distintas herramientas enmarcadas en Trabajos Finales de Grado durante los últimos años.

Dentro del marco previamente expuesto, se han desarrollado dos plataformas que sirven de base para el proyecto objetivo de este documento.

- **RoboHealth Arm** es un brazo robótico de tres grados de libertad (actualmen-

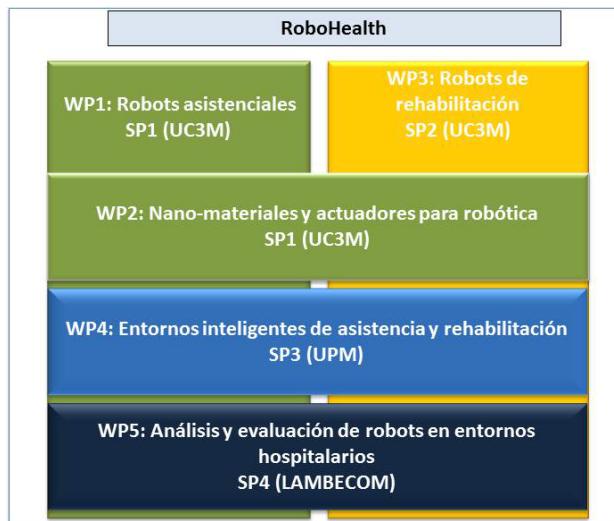


Figura 1.1: Estructura RoboHealth

te, cuenta con sólo dos grados de libertad operativos) diseñado para sustentar una tablet en su extremo, haciendo más accesible su uso para pacientes y cuidadores. Está basado en un sistema de cuerdas y poleas accionado por tres servomotores.

- Por otro lado, existe una aplicación de **Node-RED** que integra los diferentes dispositivos y proyectos desarrollados en una red domótica. Incluye una interfaz gráfica que facilita su interacción vía internet, posibilitando controlar los dispositivos desde cualquier lugar.

La idea es continuar el proceso de integración de los diferentes dispositivos en Node-RED con la intención de controlar todo desde la misma interfaz. En ese contexto, surge el proyecto de hacerlo con el brazo RoboHealth Arm. Con el fin de tratar de explorar todas las tecnologías posibles, se comprueba que la radiofrecuencia aún no había sido y existen soluciones económicas en el mercado.

Así, el planteamiento del Trabajo Final de Grado tomó forma, definiéndose como la conexión mediante el uso de radiofrecuencia del brazo RoboHealth Arm a la interfaz de Node-RED. Para la radiofrecuencia, se usarán dispositivos XBee.

1.2. Objetivos

Como se ha indicado con anterioridad, el **objetivo global** del proyecto es la completa integración de un control a través de internet del brazo robótico. Los comandos se lanzan desde la interfaz de Node-Red y el ordenador de la sala transmite la orden vía radiofrecuencia al brazo.

Posteriormente, se pueden establecer pequeños **objetivos parciales** que resulten en la consecución completa del proyecto. Estos objetivos secundarios están más

orientados al correcto funcionamiento de cada una de las etapas y tecnologías utilizadas, así como de la correcta interacción entre estas y su posterior integración. Es este el procedimiento que se ha seguido a lo largo de todo el trabajo: hacer funcionar cada etapa de manera individual, para después ir integrándolas paso a paso.

Los objetivos parciales mencionados son los siguientes, yendo desde el lado de Node-RED hacia el lado del RoboHealth Arm:

- Un programa de flujos en Node-RED debe correr sin errores en la Raspberry Pi, generando un nuevo apartado en la actual interfaz para controlar el RoboHealth Arm.
- Desarrollar una *user interface* para comandar el brazo desde Node-RED. El objetivo no es otro que permitir configurar las coordenadas articulares del brazo, parámetros necesarios en el frame que posteriormente deberá recibir el RoboHealth Arm. Una vez configurados, se tendrá acceso a un botón encargado de poner en marcha la transmisión de información.
- La orden enviada desde la interfaz de Node-RED pondrá en marcha un script programado en Python que tomará como parámetros la configuración previamente establecida y enviará por uno de los puertos serie el frame generado de acuerdo a las especificaciones de diseño de la comunicación del RoboHealth Arm y el encapsulamiento de las comunicaciones vía radiofrecuencia. Toda la gestión de la ejecución de este script ha sido programada en Node-RED.
- El firmware cargado a los dispositivos XBee debe hacerlos compatibles entre ellos, de acuerdo a las características y casos de uso a los que cada uno se va a enfrentar.
- La configuración de los módulos XBee es vital para su comunicación. Esta configuración debe habilitar la comunicación entre los dos dispositivos XBee sin dejar de permitir la comunicación serial con el RoboHealth Arm ni con la Raspberry Pi. Es decir, los módulos XBee deben ser configurados de tal manera que esta configuración sea intersección entre la compatible con el brazo robótico y la compatible con la Raspberry Pi.
- Un dispositivo XBee ha sido configurado para enviar el frame de datos recibido por comunicación serial. Al poder concentrarse todo el procesamiento de la información correspondiente al emisor en el anteriormente mencionado script, no se precisa de ningún microcontrolador adicional que funcione junto al módulo de radiofrecuencia. Así pues, el módulo XBee funciona de manera exclusiva como un traductor entre la información en el puerto serie correspondiente y las ondas de radiofrecuencia.
- El dispositivo XBee receptor de la información que comanda el brazo robótico está situado en el mismo. Su objetivo es ser capaz de captar el mensaje de radio específicamente diseñado para él y transmitirlo al microcontrolador del brazo. De la misma manera que en el otro XBee, su función será la de traductor de las ondas de radio (exclusivamente de las destinadas a él) en información en el puerto serial. Por tanto, no es necesario procesar en ningún caso la información recibida a través de radiofrecuencia, evitando un segundo microcontrolador

que escuche y adapte constantemente el módulo XBee. Esto es posible gracias al prediseño de los frames de información de acuerdo a las especificaciones y protocolos de comunicación del brazo.

- El RoboHealth Arm debe ser capaz de leer e interpretar de manera correcta la información depositada en el adecuado puerto serial. Se debe provocar la reacción esperada en el brazo, moviendo sus servos hasta las coordenadas articulares especificadas.

1.3. Materiales utilizados

Con el fin de facilitar al lector una visión global de los campos objeto del presente proyecto, a continuación se indican los componentes del mismo.

1.3.1. Componentes hardware

- **Raspberry Pi 3 Model B** (figura 1.2). Ordenador central donde corre la red domótica de toda la habitación



Figura 1.2: Raspberry Pi 3 model B

- **Arduino Uno R3 o clon** (figura 1.3). Plataforma necesaria para el uso de la XBee Shield.

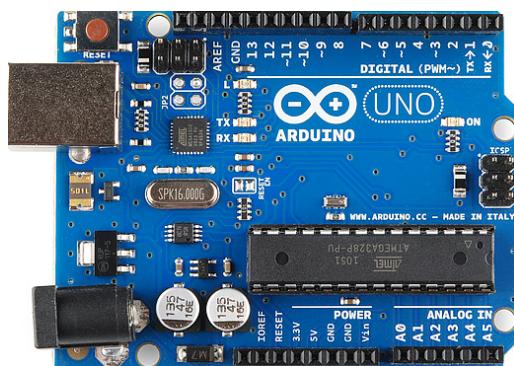


Figura 1.3: Arduino Uno R3

- **Xbee Shield** (figura 1.4 ¹). *Add-on* que permite la interacción sencilla con el módulo XBee a través de Arduino.



Figura 1.4: Xbee Shield

- **XBee S2** (figura 1.5). Módulo de radiofrecuencia para la transmisión inalámbrica de datos

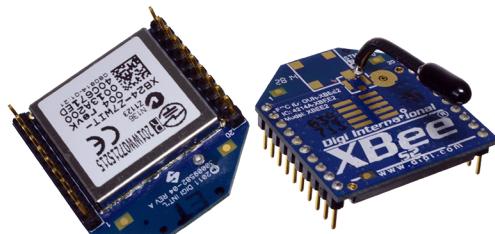


Figura 1.5: Xbee Module

- **Arduino Mega** (figura 1.6). Microcontrolador sobre el que se monta RoboHealth Arm.



Figura 1.6: Arduino Mega

¹La imagen contiene el módulo XBee además de la Xbee Shield

1.3.2. Componentes software

- **Raspbian.** Sistema operativo instalado sobre la Raspberry Pi.
- **Python script.** Programa escrito en lenguaje Phyton para ser ejecutado por la Raspberry Pi.
- **Node-RED** (figura 1.7). Aplicación programable que es capaz de integrar múltiples dispositivos hardware.

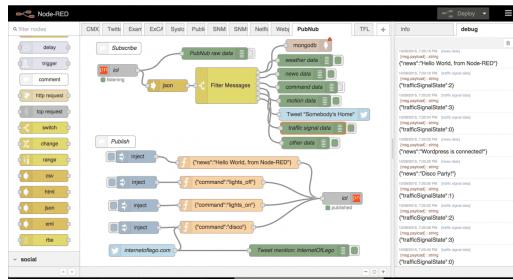


Figura 1.7: Interfaz de edición de Node-RED

- **RHA.** Software del RoboHealth Arm.

1.4. Estructura del documento

A continuación, y para facilitar la lectura del documento, se detalla el contenido de cada capítulo.

- En el capítulo 1 se realiza una introducción al proyecto. Un breve comentario sobre la idea y componentes del trabajo.
- En el capítulo 2 se exponen los fundamentos teóricos que pueden facilitar la lectura posterior del desarrollo del proyecto.
- En el capítulo 3 se encuentra el estado del arte, un repaso a la tecnología actualmente desarrollada incluida en el proyecto, para conocer con mayor precisión el punto de partida del mismo.
- En el capítulo 4 se detalla el desarrollo del trabajo. Se exploran las soluciones hardware, software, montaje...
- En el capítulo 5 se exponen y discuten las pruebas y los resultados del proyecto.
- En el capítulo 6 se describe la gestión del proyecto; incluyendo la planificación, el presupuesto, ciclo de vida...
- Para finalizar, en el capítulo 7 se termina con las conclusiones sacadas del proyecto y potenciales desarrollos futuros.

Capítulo 2

Marco Teórico

El marco teórico del proyecto se limita al estudio de la naturaleza de las comunicaciones usadas. El trabajo emplea dos tipos de comunicaciones: radiofrecuencia y serial. La radiofrecuencia es la comunicación usada entre los módulos XBee. Por otro lado, la comunicación entre los módulos mencionados y sus correspondientes dispositivos de control se realiza por método serial. A continuación se comentan los conceptos básicos para comprender ambos métodos de comunicación.

2.1. Conceptos de la comunicación serial

La comunicación serie (o serial) es un método de transmisión de datos consistente en el envío de un único bit en un mismo instante de forma secuencial por una simple línea de transmisión. Lo simple de este método ha hecho que la comunicación serial se extienda masivamente entre los dispositivos comerciales, siendo actualmente un método común para comunicar ordenadores con distintos periféricos.

Se opone a la llamada comunicación paralela, que precisa de una línea de transmisión por cada bit de datos a cambio de un aumento de las prestaciones. Es bastante usual usar ocho líneas de datos, correspondiente a un byte. El uso de un número elevado de líneas de datos además de las líneas de control, hace notablemente mas caro el uso de comunicación paralela. Sumado a esto, la comunicación serie se ha desarrollado bajo un marco de estandarización mucho más extendido que en la comunicación paralela [4]; yendo esta característica en contra de implantar sistemas paralelos.

2.1.1. Características

Los bits secuenciales son transmitidos a partir del uso de dos niveles lógicos que pueden ser de tensión (más usual) o de corriente. Estos niveles se corresponden en el llamado *formato marca espacio* (figura 2.1) con los niveles lógicos "0" y "1". Al nivel lógico "0" se le denomina espacio mientras que al nivel lógico "1" se le llama marca.

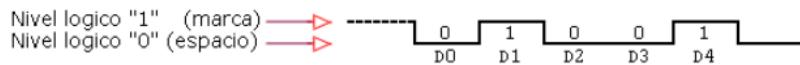


Figura 2.1: Formato serie marca/espacio

Existen varias configuraciones de las líneas de comunicación [4], cada uno optimiza sus prestaciones en relación a los requerimientos de la aplicación.

- La configuración *simplex* únicamente precisa de un hilo de comunicación y, pese a su ligero menor coste de producción, no es muy usado debido a sus limitaciones. Estas limitaciones son, en primer lugar, la escasa flexibilidad resultado de la necesidad de establecer una relación emisor-receptor permanente; es decir, uno de los dispositivos será siempre emisor y el otro será siempre receptor. Por otro lado, no se permite la comprobación de la recepción de la información, posibilitando comunicaciones deficientes.
- La configuración *semi duplex* utiliza igualmente una línea de comunicación pero conmutando la etiqueta de emisor y receptor entre los dispositivos de manera periódica. Uno de los dispositivos emite la información para pasar a recibir la a continuación. Esto soluciona los problemas de la configuración *simplex*, permitiendo hacer una comprobación de errores en la transmisión de datos.
- La mayoría de casos de uso de la comunicación serie emplean la configuración *full duplex* que permite la transmisión y recepción simultánea de datos por parte de ambos dispositivos. Para ello, se hace uso de dos líneas de comunicación, una destinada a la transmisión y otra a la recepción.

La base del funcionamiento de la comunicación es la sincronización. Para resolver esta cuestión en el caso de la comunicación serie, se estandarizan varios bits (o series de bits) y parámetros entre emisor y receptor con el fin de determinar cuál es la información.

Existen dos modos de transmisión:

- El modo **síncrono** no usa bits de sincronización y todos los componentes de la transmisión se agrupan en bloques consecutivos, existiendo una secuencia de sincronización al inicio de cada bloque. El emisor indica al receptor que se va a iniciar una comunicación mediante el envío de un octeto de bits "sync" (figura 2.2).

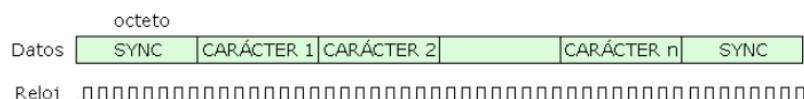


Figura 2.2: Transmisión serial síncrona

- Por otro lado, el modo **asíncrono** no utiliza una línea de reloj, por lo que deben coincidir las características de la comunicación, como la velocidad de

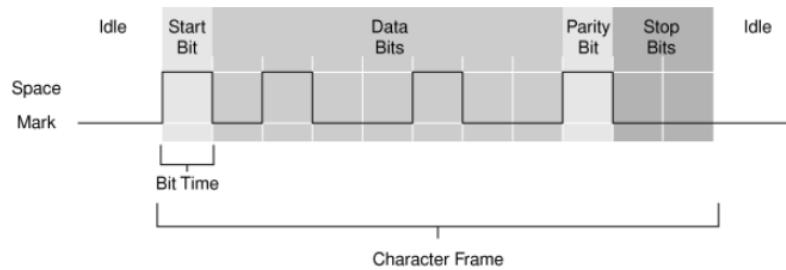


Figura 2.3: Transmisión serial asincrónica

transmisión de datos, entre los dispositivos conectados. Para conseguir la sincronización en la comunicación, se hace uso de, por ejemplo, bits de inicio o parada (figura 2.3). Hay que tener en cuenta que la frecuencia a la que estos dispositivos leen el estado del frame es mucho mayor que la frecuencia a la que cambian de estado los propios bits del frame. En las siguientes líneas se definen conceptos relacionados con este tipo de comunicación, que es la utilizada a lo largo del proyecto.

La indicación del inicio de la comunicación del frame transmitido se realiza mediante el **start bit**. Este bit genera un flanco negativo (transición de nivel marca a nivel espacio) cuando la linea de datos está a nivel marca ("1" lógico) mientras no se transmite información (figura 2.3).

Existen varios parámetros que especifican y definen la comunicación serial [15], y que deberán ser comunes entre los dispositivos partícipes de la transmisión de datos. A continuación se enumeran los principales parámetros configurables. Una diferencia en la configuración de los dispositivos impedirá su comunicación.

- La llamada **Baud rate** o, en castellano, tasa de baudios, se trata de la cuantificación de las símbolos por segundo de una comunicación y sirve para medir la velocidad de transmisión de la información. Coincide con los bits por segundo siempre que un símbolo de transmisión contenga un bit. Sin embargo, esto no tiene porqué cumplirse y el número de bits transmitidos por segundo pueden ser mayores que los baudios. La velocidad de transmisión se limita con el ancho de banda y la potencia de la señal.
- El número de **bits de datos** que se precisan para codificar un símbolo de información transmitido. Se suele tender a estandarizar el número de bits entre 5 y 8 bits.
- El **Parity bit** es opcional y se usa para la detección de errores. Se incrusta en medio del frame de bits de tal modo que es comprobable la correcta recepción del mismo. Se puede configurar como un bit de paridad par o impar. En el primer caso tendrá el valor necesario para hacer que la suma de los bits en nivel lógico "1" correspondiente a los datos y al propio bit de paridad sea par. En el segundo caso ocurre lo contrario, el número de bits en estado alto debe ser impar.

- Los llamados **Stop bits** son bits que emplean una tensión positiva para informar de la finalización del frame hasta el siguiente flanco negativo. Suelen ser uno o dos los bits de paradas

2.1.2. Problemas

Los defectos más notables relacionados con la comunicación serial vienen de a mano de la sincronización y de la pérdida de bits.

La sincronización se consigue haciendo que los dispositivos conectados "hablen el mismo idioma". Para ello, hay que hacer coincidir toda una ristra de parámetros, cuyos principales exponentes han sido comentados previamente. Cualquier tipo de discrepancia hará que se pierda información o, si la información perdida viene relacionada con los delimitadores de la información comunicada, no se pueda producir la transmisión de datos.

En cuanto a la pérdida de información, existen mecanismos que posibilitan su detección y permiten actuar en consecuencia (por ejemplo, solicitando un nuevo envío de la información).

- Los **generadores y detectores de paridad** comparan el bit de paridad con la información enviada, tanto en la emisión de los datos como en la recepción de los mismos. Si existen incoherencias, se genera un bit de error que, posteriormente, es usado para actuar en consecuencia. La paridad, como se ha detallado previamente, puede ser par o impar y este marco debe ser común a lo largo de toda la comprobación del error.
- El método **checksum** soluciona de manera sencilla la potencial circunstancia en la que dos bits se transmiten de manera errónea y compensan mutuamente su error, siendo indetectables mediante el método de paridad. El checksum se trata de un bit añadido al final de la transmisión que contiene la información resultante del complemento a dos de la suma binaria de todos los bytes transmitidos en el mensaje. El receptor sumará los bytes recibidos, incluyendo el checksum, y el resultado debería ser cero si la comunicación ha sido la correcta.

2.1.3. Usos y aplicaciones

La comunicación serial es ampliamente utilizada en la comunicación de ordenadores con sus periféricos a través de los puertos USB. Existen una serie de estándares de comunicación serial. Los principales son los siguientes:

- El **enlace TTL** utiliza los típicos niveles de 0 y 5 voltios para definir sus estados lógicos. No es recomendable para la transmisión de datos a medias y largas distancias (no a más de 5 metros).

- El **lazo de 20mA** tiene la particularidad de funcionar con niveles de intensidad. El nivel lógico de marca se logra con 20mA, mientras que la ausencia de corriente corresponde al nivel de espacio. El trabajar a intensidad permite la comunicación a largas distancias (no superiores a 1.6 kilómetros).
- Una de las normas que regulan el uso de la comunicación serial más utilizadas a lo largo de la historia es la **RS232**. Se trata de un protocolo de comunicación serie desarrollado a lo largo de los 70 que fue implementado en los ordenadores de la época. Hoy en día, ha sido ampliamente superado por la conexión USB, iniciando su declive.

La tensión de funcionamiento oscila entre los 12V y el mismo valor negativo. Existe un rango entre 3 y -3 voltios que está restringido al uso por comunicación serial RS232 y que absorbe errores de comunicación o ruido, evitando caer en indeterminaciones en la señal.

2.2. Conceptos de la comunicación por radiofrecuencia

La comunicación por radiofrecuencia hace uso de ondas de radio para transmitir información a distancia. Estas ondas se basan en la interacción de campos eléctricos y magnéticos.

2.2.1. Características

Las ondas de radio son un tipo de ondas electromagnéticas cuyas longitudes de onda son mayores que la luz infrarroja. El espectro de las longitudes de onda de las ondas de radio se sitúa entre los 100 micrómetros y los 100 kilómetros. Es interesante mencionar que la naturaleza produce ondas de este tipo mediante fenómenos como el rayo por lo que, en su generación artificial, es importante aislar la comunicación del ruido externo. La naturaleza de estas ondas hacen que sus propiedades físicas sean variables con la frecuencia; en función de la aplicación, será más conveniente una frecuencia u otra.

El transmisor de radio (figura 2.4) es un dispositivo electrónico cuyo fin es tomar una señal a enviar, codificarla (modularla) y emitirla en forma de onda electromagnética por una antena. Por su parte, un receptor de radio se encarga de aprovechar la inducción electromagnética producida por las ondas de radio amplificándola y decodificandola de acuerdo al procedimiento seguido por el emisor. La forma de la transmisión, preacordada entre emisor y receptor, es vital para facilitar la distinción entre señal y ruido.

Las transmisión de ondas de radio se basa en la modificación de una onda base de acuerdo a la señal que se desea transmitir. Este proceso se denomina **modulación**. Este mecanismo de modulación maximiza la cantidad de información transmitible de manera simultánea, a la vez que incrementa la robustez haciendo al sistema más resistente a ruidos, interferencias y perturbaciones. El proceso opuesto a la modu-

lación es la llamada demodulación cuyo fin es la obtención de la señal transmitida previamente y suele ser realizada por el mismo receptor de la información.

La onda base a la que se ha hecho referencia previamente se denomina **onda portadora**. Usualmente se trata de una onda sinusoidal que puede ser modificada en alguno de sus parámetros durante el proceso de modulación previo a la transmisión [16].

Existen numerosas técnicas de modulación. Algunas forman parte del lenguaje popular debido a su extendido uso y otras, de un desarrollo más reciente, tienen aplicaciones más específicas. En función del parámetro sobre el que se actúe, se puede dar con diferentes tipos de modulación. A continuación se listan algunos y se comentan aquellos que tienen un uso más extendido o que tienen aplicación en el actual proyecto.

Técnicas de modulación analógica:

- La **amplitud modulada**, comúnmente denominada AM, es una técnica que incide en la modificación de la amplitud de la onda portadora (figura 2.5). El resultado es una onda de igual frecuencia que la onda portadora pero que ve su amplitud variable a lo largo del tiempo en función de la señal a transmitir. Su simpleza hizo que fuera el primer método usado para tener éxito en la transmisión de audio vía telefónica.
- La **frecuencia modulada**, o FM, se trata de una técnica de modulación focalizada en la variación de la frecuencia de la onda portadora [14] (figura 2.5). Si hablamos de aplicaciones analógicas, tras la modulación se obtiene una onda cuyo valor de la frecuencia es proporcional a la señal a transmitir. Es frecuente emplear un condensador variable denominado varactor (junto a un cristal piezoelectrónico) que varíe ligeramente la frecuencia del oscilador en función de la señal a transmitir.
- La **modulación de fase**, también denominada PM, modifica de manera proporcional la fase de la onda portadora con la señal moduladora (figura 2.6). Es menos utilizada que las anteriores porque presenta ciertos problemas de am-

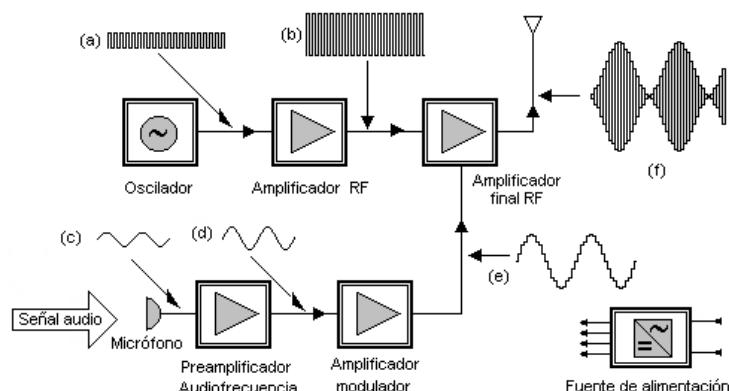


Figura 2.4: Ejemplo de radiotransmisor AM

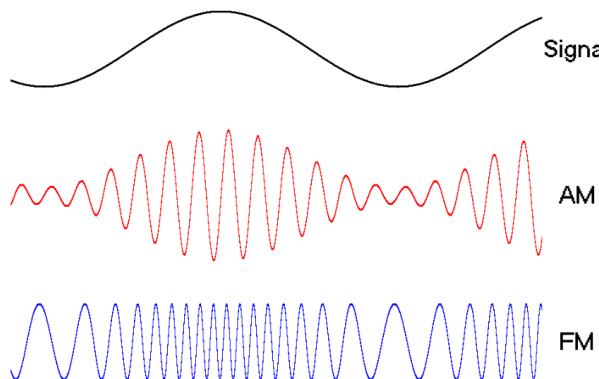


Figura 2.5: Modulación de la señal

bigüedad en los extremos del rango de modulación y, especialmente, porque el coste y la complejidad de los equipos de recepción requeridos es superior.

- Modulación de amplitud en cuadratura [14], o QAM.
- Modulación de doble banda lateral [14], o DSB.
- Modulación de banda lateral única [14], o SSB.

Técnicas de modulación digital:

- La **modulación por desplazamiento de fase**, también denominada por sus siglas en inglés PSK, consiste en la variación de la fase de la onda portadora entre una determinada variedad de valores discretos. Es, en resumen, la técnica análoga a la PM pero con la particularidad de usar una salida discreta con un número limitado de estados.
- Modulación por desplazamiento de amplitud, o ASK.
- Modulación por desplazamiento de frecuencia, o FSK.

La técnica de modulación de espectro ensanchado se basa en la expansión de la señal a transmitir a lo largo de una banda muy ancha de frecuencias. Lógicamente, este método no es el más eficiente en cuanto al uso del ancho de banda pero es combinable con otros métodos que hagan uso de un ancho de banda mucho más estrecho. El receptor, al interpretar la información que llega, va a ver su ruido muy ligeramente incrementado al coexistir con estas otras señales debido a que se dedicará a escuchar un ancho de banda muy amplio. Existen varias técnicas principales de ensanchado del espectro:

- El ensanchamiento de espectro por secuencia directa (figura 2.7), o DSSS, incrusta un patrón de bits (pseudoruido) redundante entre cada uno de los bits que componen la señal a transmitir. Cuanto mayor sea este patrón de bits,

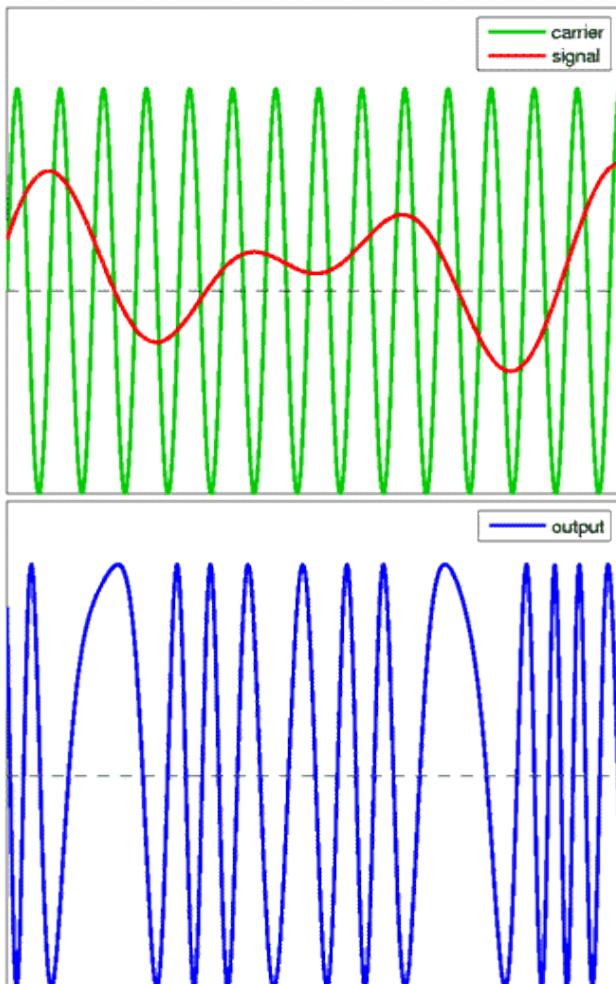


Figura 2.6: Modulación de fase

más se parece la señal modulada al ruido y, consecuentemente, más interpretable como tal será por los dispositivos a los que no se dirija la información. La resistencia a interferencias es proporcional al tamaño del patron de bits. La secuencia de bits que se utiliza para modular la señal se conoce como **secuencia de Barker** y deberá ser conocida por el receptor para poder demodular la señal y obtener la información. Se han estandarizado dos tipos de modulación para la técnica de espectro ensanchado por secuencia directa. Una de ellas es la modulación de fase binaria (DBPSK) y la otra es la modulación de fase por cuadratura en offset (OQPSK). Ambas técnicas de modulación son casos específicos de la modulación por desplazamiento de fase mencionada previamente. Los módulos XBee usados en el proyecto trabajan bajo esta estandarización.

- Ensanchamiento de espectro por salto de frecuencia, o FHSS.

Como sucedía en el caso de la comunicación serial, en cuanto a la transmisión vía radio frecuencia también se puede hablar de diferentes configuraciones de la línea de comunicación [11]. De forma análoga, uno se puede encontrar con las siguientes configuraciones:

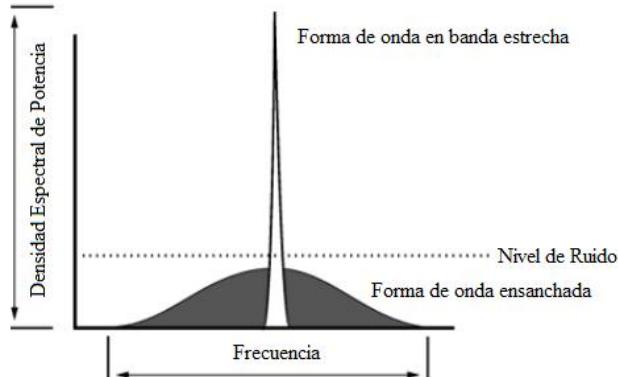


Figura 2.7: Banda estrecha vs DSSS

- El modo de comunicación **simplex a una frecuencia** (figura 2.8) consta de un emisor al que todos los equipos receptores están escuchando. Es barato pero puede ocasionar problemas de interferencias o de captura de comunicación.

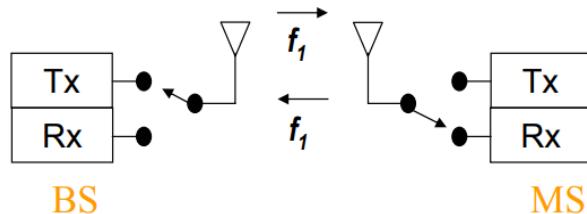


Figura 2.8: Radiocomunicación simplex a una frecuencia

- El modo **simplex a dos frecuencias** (figura 2.9) trata de evitar la interferencia entre dos dispositivos emisores cercanos.

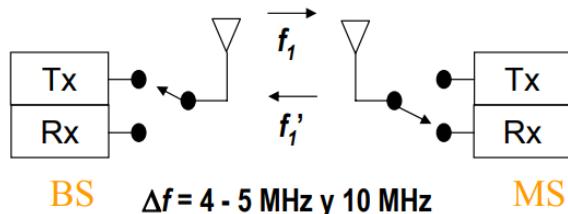


Figura 2.9: Radiocomunicación simplex a dos frecuencias

- El modo **semiduplex** (figura 2.10) usa un duplexer para retransmitir hacia los receptores lo que recibe de otro emisor.

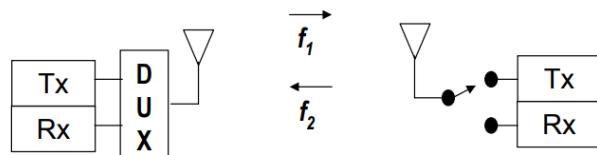


Figura 2.10: Radiocomunicación semiduplex

- El modo **duplex** (figura 2.11) emplea un duplexor para cada dispositivo con el fin de que todos los dispositivos funcionen de emisores y receptores con el contra de elevar el coste

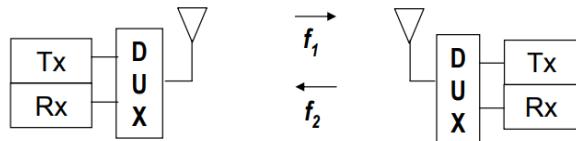


Figura 2.11: Radiocomunicación duplex

2.2.2. Problemas

De igual manera que en cuanto a la comunicación serial el principal problema venía de la sincronización de la información, en el caso de la radiofrecuencia se puede afirmar que el principal problema proviene de las interferencias y el ruido.

Al igual que otros aspectos de la radiocomunicación como el alcance o la potencia son adaptables, existiendo un amplio rango de elección y habiendo un importante número de dispositivos muy versátiles en el mercado; el ruido electromagnético es una cuestión inevitable porque su naturaleza es igual a la de la información recibida y su existencia es inherente al entorno. Aún así, existen formas de reducir al mínimo la influencia de este ruido.

El elemento de la antena purifica la transmisión de la señal. El uso de una alta frecuencia portadora limita su coste y tamaño manteniendo un diseño adecuado.

Varios métodos de modulación, en especial aquellos relacionados con la modulación en frecuencia, tienden a ser más inmunes a la interferencia y al ruido. Esto es debido a que un patrón variado de frecuencias conocidas es más fácil de distinguir del ruido que una frecuencia permanente durante toda la transmisión. Ahí tenemos la explicación a por qué la escucha de radio FM es usualmente más estable y de mayor calidad que la radio AM. La contra partida a esto es que la eficiencia del ancho de banda se reduce.

Por otro lado, en la etapa de modulación, se viene comprobando que, en general, conforme los procesos de transmisión y modulación se vuelven más complejos o incluyen un aumento de la velocidad de transmisión de los datos; se pierde rendimiento de la comunicación, tanto a nivel de inmunidad ante las perturbaciones, como en cuanto a la cobertura [16].

2.2.3. Usos y aplicaciones

La versatilidad de la radiocomunicación ha hecho que su uso se expanda en multitud de campos. A lo largo de su extensa historia, las aplicaciones han sido variadas y a continuación se listan algunas de las más representativas:

- Quizás la primera aplicación de las ondas de radio fue el establecimiento de redes de radioayuda que permitieran el envío de información en el conocido código morse, especialmente en el ámbito naval. Hoy en día también se implementa en la aeronavegación
- La radio, como medio de comunicación desde que se implementara en Buenos Aires, lleva más de un siglo funcionando [17]. Su influencia en el desarrollo del siglo XX es incalculable.
- El heredero como rey de los medios de comunicación de la radio fue la televisión y también usaba esta tecnología. Hasta hace escasos años la señal de televisión se hacía llegar a las casas a través de ondas analógicas de radio que ocupaban las bandas VHF y UHF.
- Multitud de radioaficionados y comunidades usan esta tecnología como medio de comunicación independiente a nivel local.
- Las redes inalámbricas se han vuelto recientemente muy populares al mismo tiempo que usando, entre otros, radiofrecuencia, han ido sustituyendo a los cables. El presente proyecto viene a ser una aplicación específica de este uso.
- Servicios de transmisión de audio y vídeo
- Telefonía móvil

Capítulo 3

Estado del arte

A continuación, se hace un repaso de la tecnología existente en la actualidad relacionada con los campos tocados por el presente proyecto. El objetivo es proporcionar al lector un punto de partida del trabajo y comentar las soluciones utilizadas por otros autores.

3.1. Water level control using Raspberry Pi + XBee + XBMQ + MQTT + Node-Red [9]

Se trata de un proyecto que trabaja de manera bastante similar al trabajo objetivo del presente documento, teniendo ciertas tecnologías en común.

El objetivo es el control de una pequeña bomba de agua que llena un depósito lentamente, con una distancia considerable entre ambos elementos y el ordenador central. La motivación para automatizar el proceso de llenado del depósito era evitar ciertos incidentes provocados por el olvido de la persona que controlaba la bomba, debido a los extensos tiempos que se tomaba el sistema para llenar el depósito. El autor comenta que la señal se perdía frecuentemente al usar unos módulos Wi-Fi baratos y terminó decidiéndose por la tecnología XBee de Digi, descartando cualquier tecnología que no fuera inalámbrica por los motivos de distancia comentados previamente.

Una Raspberry Pi3 ejerce de ordenador central del sistema y es el componente encargado de coordinar la interacción con el usuario que manda la orden y la interacción con el depósito y la bomba de agua vía XBee (montaje en la figura 3.1).

- Por un lado, tiene instalado Mosquitto (MQTT); un programa que usa la red para crear una especie de "tablón de anuncios" virtual. Los dispositivos conectados a esa red pueden suscribirse a un *topic* y recibir lo que otros dispositivos publican en él. La idea es que, haciendo uso de diferentes aplicaciones (como puede ser el caso de *IoT MQTT Dashboard* en Android), se puedan publicar ciertos mensajes en algún *topic* al que esté suscrita la Raspberry Pi desde otros

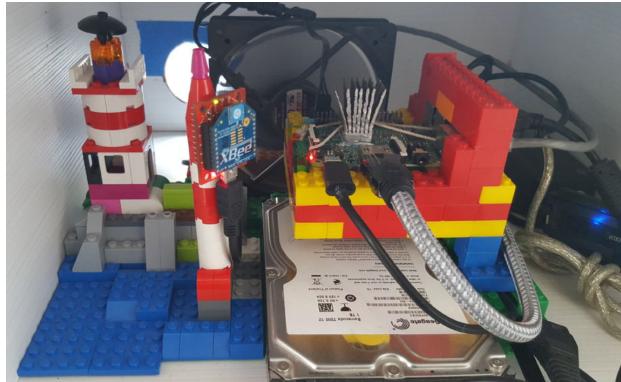


Figura 3.1: Montaje del proyecto 3.1

dispositivos de manera remota. La Raspberry recibe esta información y actúa en consecuencia.

- XBMQ actúa de puente dentro de la Raspberry Pi entre MQTT y el módulo XBee conectado a la misma, que es el que se comunica con el actuador y el sensor.
- La Raspberry Pi envía los comandos al actuador y recibe la información del sensor a través del módulo XBee. En el lado del depósito, se sitúa el otro XBee con una salida controlando un pequeño relé que acciona y desactiva la bomba y una entrada que envía al XBee de la Raspberry el estado del sensor. Huelga mencionar que ambos XBee se han debido configurar adecuadamente.
- Para detener la bomba si el sensor detecta que el depósito está lleno, se precisa de cierta lógica que se programa en la Raspberry usando otra tecnología que se verá mas en detalle: Node-Red. El uso de este programa permite automatizar otras acciones paralelas como, por ejemplo, publicar un tweet cada vez que la bomba cambie de estado.

3.2. Servidor Raspberry Pi-XBee en Python [3]

El proyecto detalla la conexión de un Arduino Uno y una Raspberry Pi a través de XBee. Viene siendo algo muy similar a una de las etapas del trabajo detallado en este documento.

En el lado del Arduino Uno, emplea una XBee Shield 2.0 de Sheeед Studio (figura 3.2) para implementar el módulo de radio frecuencia.

Si se habla del lado de la Raspberry Pi, la interacción con el módulo XBee se realiza a través de una placa XBee Explorer.

Como uno puede comprobar, existe una gran variedad de placas y shields que adaptan y facilitan la interacción de los módulos XBee con las plataformas de ordenadores y microcontroladores más populares.

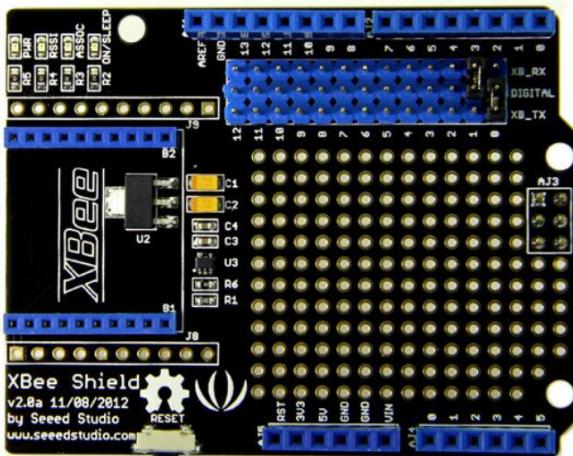


Figura 3.2: XBee Shield v2.0 - Seeed Studio

En este caso, los módulos XBee están configurados como AT. Más adelante en el documento se detalla esta cualidad.

3.3. Smart Porch Light Project [12]

En el contexto del Internet de las Cosas (IoT), el autor desarrolla un proyecto de red inalámbrica de sensores y actuadores sincronizados a una nube en la red.

El caso particular consta de una luz de exterior que es automáticamente controlada teniendo en cuenta los datos obtenidos de múltiples sensores que captan el estado del entorno. Estos sensores miden la luz ambiente, temperatura, humedad y luz ultravioleta. El control de la lámpara de exterior se lleva a cabo con un relé capaz de separar la etapa de potencia de la etapa de señal. Con el proyecto operativo, se usan servicios en la nube para almacenar y leer la información del estado del sistema; obteniendo como resultado una lámpara inteligente que permite mostrar a los usuarios el estado de los sensores desde cualquier dispositivo con acceso a internet.

Los componentes hardware, al ser todos del mismo fabricante (Seeed Studio), permiten su apilamiento en un único stack compacto (figura 3.3) de microcontrolador, shields módulo XBee. El microcontrolador es un Arduino Mega, los shields son de sensores y de XBee y, coronando el stack, se encuentra el módulo XBee LTE.

A diferencia de proyectos mencionados anteriormente, en este se usa una API REST en lugar de MQTT para transferir la información al servicio de nube correspondiente y los módulos XBee son del modelo LTE, por lo que pueden acceder a la red directamente sin la necesidad de poseer enlace alguno a un ordenador con conexión, bien vía USB o a través de otro módulo XBee. Por otro lado, la interfaz donde mostrar el estado de los sensores no se basa en Node-RED, sino en Ubidots.

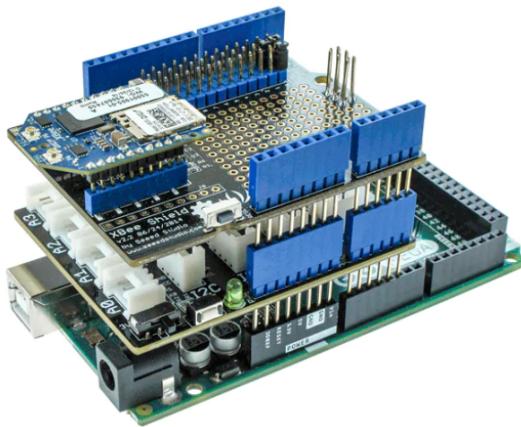


Figura 3.3: Stack Hardware del proyecto 3.4

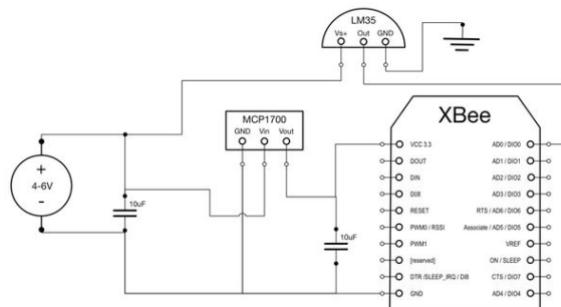


Figura 3.4: Esquema de conexión del sensor de temperatura

3.4. Home Automation System [5]

Estamos ante otro proyecto del campo IoT que hace uso de módulos XBee. En este caso, los actuadores y sensores de una casa en la que se ha implementado este sistema domótico están conectados a red inalámbrica de módulos XBee mediante cableados como el que se muestra en la figura 3.4.

La información se transmite a un portal central que tiene como base una placa Netduino o una Raspberry Pi con Windows IoT como sistema operativo. El portal envía y recibe mensajes de cada uno de los módulos XBee conectados y traduce la información de tal manera que el controlador lo pueda interpretar.

Usa MQTT para la comunicación controlador-portal pero recurre a una nueva herramienta, openHAB, para proporcionar una interfaz gráfica donde monitorizar y controlar los sensores y actuadores de la casa.

3.5. CONTROLADOR CENTRAL PARA UN SISTEMA DOMÓTICO UTILIZANDO EL PROTOCOLO INALÁMBRICO ZIGBEE

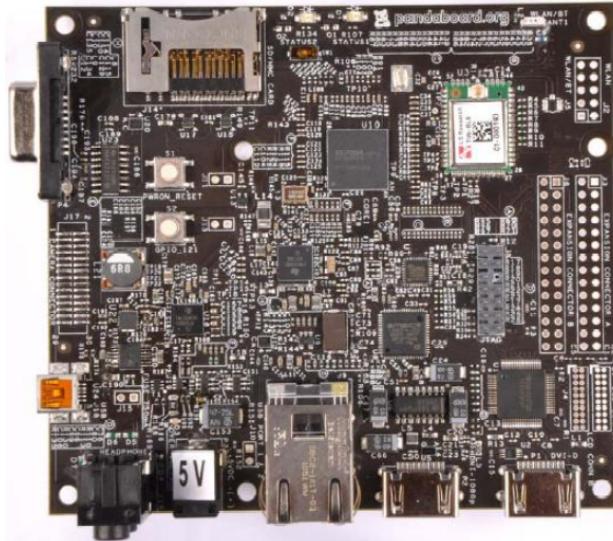


Figura 3.5: PandaBoard ES

3.5. Controlador central para un sistema domótico utilizando el protocolo inalámbrico ZigBee [2]

Con este proyecto se habla del desarrollo de otro sistema domótico; esta vez basado en la placa de desarrollo Pandaboard (figura 3.5). A principios de la presente década, Pandaboard nació para competir en el mercado de los ordenadores pequeños de bajo coste, pero Raspberry terminó por dominar a la competencia. Hoy en día se trata de una placa obsoleta y poco usada por la comunidad.

En este proyecto se recurre también al uso de dispositivos XBee para la transmisión de información y a MQTT. Incorpora Domoticz, un programa de supervisión y configuración domótica de código abierto.

Como particularidad de la que se hablará de manera más detallada más adelante, en este caso los módulos XBee funcionan en configuración API, en contraposición al modo AT mencionado previamente en otro proyecto.

3.6. Node-RED based custom full-room wake-up light [6]

Proyecto que desarrolla una aplicación Node-RED con el fin de configurar un patrón despertador configurable. El desarrollo del proyecto referenciado [6] se centra en la definición de los flujos de Node-RED puesto que el sistema domótico se ha desarrollado previamente [10].

El flujo de Node-RED (figura 3.6) trabaja detectando la igualdad entre la hora programada y la actual. A continuación, comprueba si es fin de semana y si está activada la alarma durante los fines de semana en la configuración. Por último y habiendo superado las etapas anteriores, se define una secuencia de iluminación de

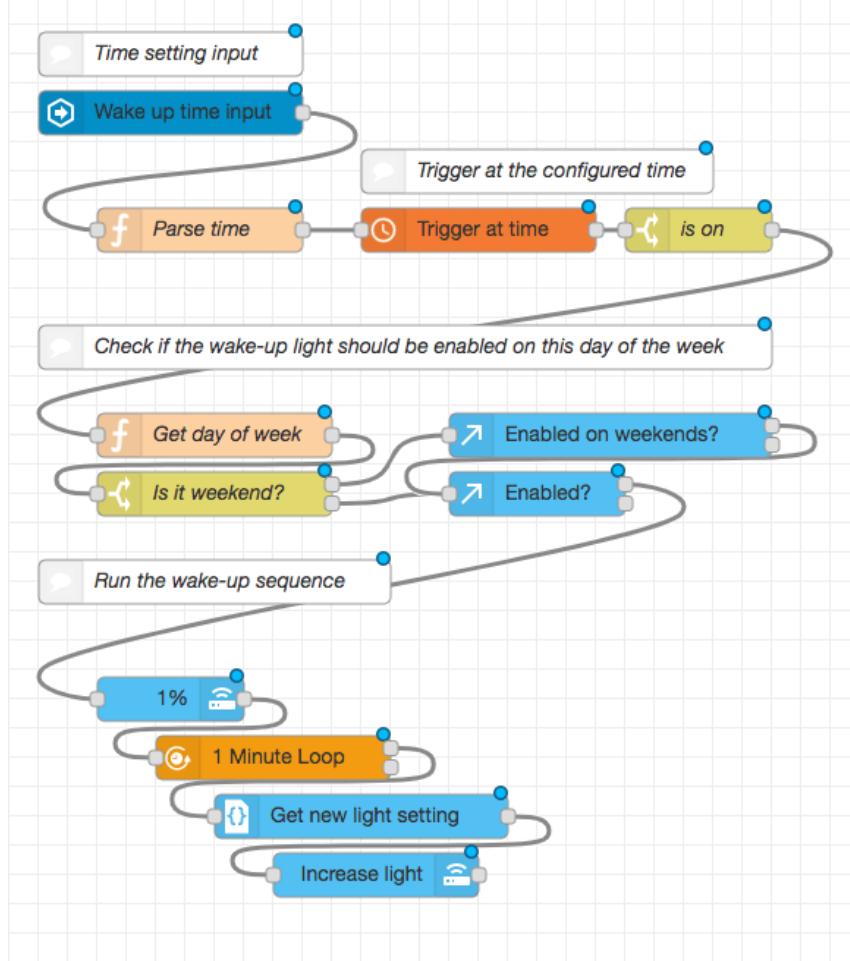


Figura 3.6: Flujo de Node-RED

las bombillas correspondiente a la señal de despertador.

El proyecto domótico completo se basa en una instalación de Node-RED y Home Assistant, que proporciona la interfaz de usuario, sobre una Raspberry Pi. Las bombillas son bombillas inteligentes que reciben órdenes vía XBee. La recepción y envío de radiofrecuencia desde la Raspberry Pi se produce a través de un hub comercial que va alternando la conexión con las diferentes bombillas a muy alta velocidad.

3.7. ArduSmartHome [1]

El proyecto ArduSmartHome consiste en el desarrollo de un sistema domótico de control que capture datos del entorno a través de varios sensores y monitorizar esa información usando Node-RED.

La principal diferencia que tiene este proyecto con los mencionados previamente es la tecnología usada para la transmisión de la información. Mientras que hasta



Figura 3.7: Arduino Yun

ahora se habría usado principalmente radiofrecuencia a través de módulos XBee¹, en este caso se usa una red WiFi local.

Para conseguir esto, se hace uso de un microcontrolador diseñado para esta tarea, el Arduino Yun (figura 3.7). La red domótica posee varios de estos microcontroladores volcando los datos de los sensores que tienen conectados a la red local. Existe un Arduino Yun² que ejerce las funciones de coordinador de las comunicaciones a la vez que es quién se encarga de establecerlas. En este proyecto también se hace uso de MQTT.

3.8. University of Minnesota – Solar Vehicle [13]

Estudiantes de la Universidad de Minnesota llevan desde 1990 desarrollado prototipos de coches solares para competir en varios certámenes a nivel nacional e internacional, obteniendo grandes resultados. En este contexto, en 2019 se ha presentado el nuevo prototipo denominado EOS II (figura 3.8).

Con el apoyo de Digi, han implementado una red inalámbrica a la que conectan ordenadores y diferentes sensores. La conexión a esta red se realiza mediante módulos XBee. El objetivo final es la comunicación entre los módulos para la detección de errores y el almacenamiento de los datos adquiridos por esos mismos sensores durante el funcionamiento del prototipo.

¹La tecnología utilizada en el proyecto que describe este documento es, precisamente, radiofrecuencia usando módulos XBee

²Se puede lograr una equivalencia económica al Arduino Yun usando un Arduino UNO complementado con la shield de extensión Dragino Yun Shield v2.4



Figura 3.8: EOS II Solar car

Capítulo 4

Desarrollo del proyecto

4.1. Planteamiento inicial

El desarrollo se ha llevado a cabo de manera modular. Se ha planteado el trabajo en bloques individuales para terminar comunicándolos por su respectiva vía.

Los objetivos descritos en la sección 1.2 sirven de guía para definir responsabilidades de cada módulo.

En la figura 4.1 se puede observar el diagrama de interacciones entre los diferentes módulos

Se establece una comunicación bidireccional en la que, por un lado, se envían comandos desde varias fuentes de mando y, por el otro, se reciben los mensajes periódicos de estado que envía el RoboHealth Arm.

Las fuentes de mando se emplazan en la red, desde donde se interactúa con el usuario. Una de ellas es Node-RED, la base de la red domótica y la otra es Mosquitto (MQTT), que diversifica el tipo de dispositivos desde los que se puede interactuar con el brazo.

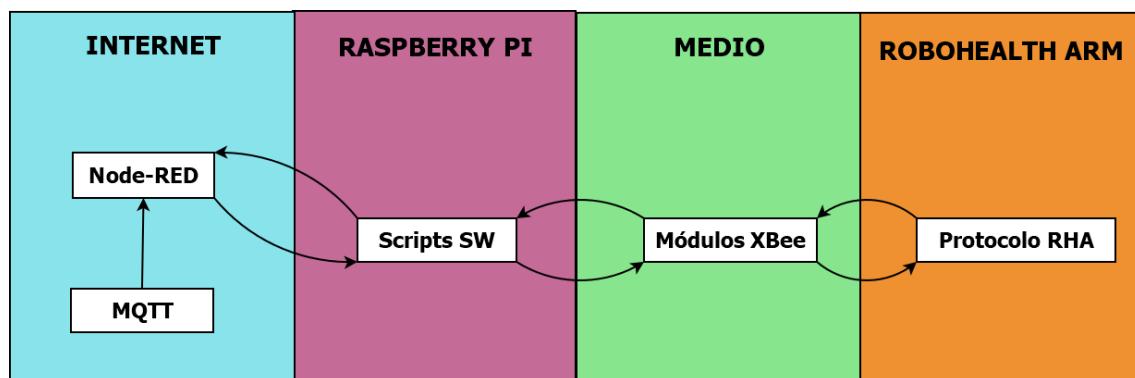


Figura 4.1: Diagrama de interacción

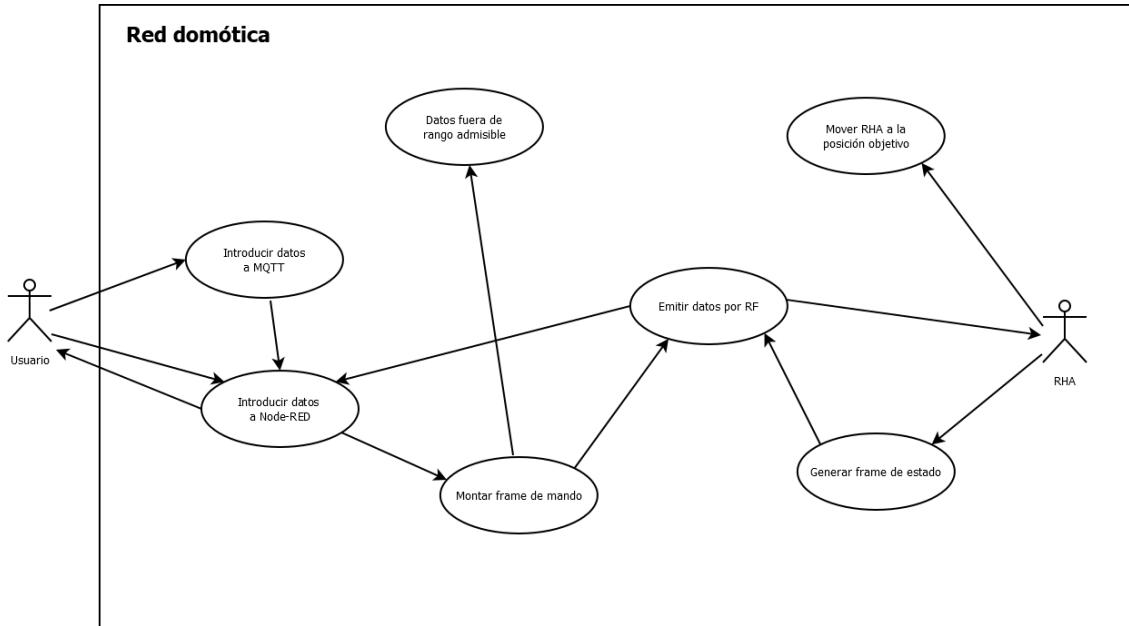


Figura 4.2: Diagrama de casos de uso

La Raspberry Pi es el punto donde las órdenes enviadas desde la red toman forma en una salida serial.

Los módulos XBee se encargan de la transmisión inalámbrica de la información a través del medio¹.

Esta información es captada por el brazo robótico de acuerdo a un protocolo de comunicación prediseñado y actúa en consecuencia.

Haciendo una analogía con los diagramas de casos de uso propios del Lenguaje de Modelado Unificado (UML) en el campo del desarrollo software, a continuación (figura 4.2) se analizan las distintas acciones que puede efectuar el usuario y cómo debería reaccionar el sistema ante estas acciones. Nótese que se ha tomado al brazo robótico como un actor más dentro del sistema domótico.

Continuando con el uso de conceptos UML, en la imagen 4.3 se puede observar la secuencia de acciones planificadas para el envío de información al brazo robótico.

Con los objetivos y conceptos establecidos, se puede pasar al desarrollo siguiendo una metodología modular, como se ha indicado antes.

4.2. Unidad de mando

La base de las comunicaciones debe ser un ordenador. Las funciones de este ordenador o unidad de mando pasan por la coordinación de los diferentes elementos de la red domótica, incluyendo tanto el control de estos como la recepción de sus

¹En la sección 2.2 se detalla el proceso del envío de ondas electromagnéticas a través del aire

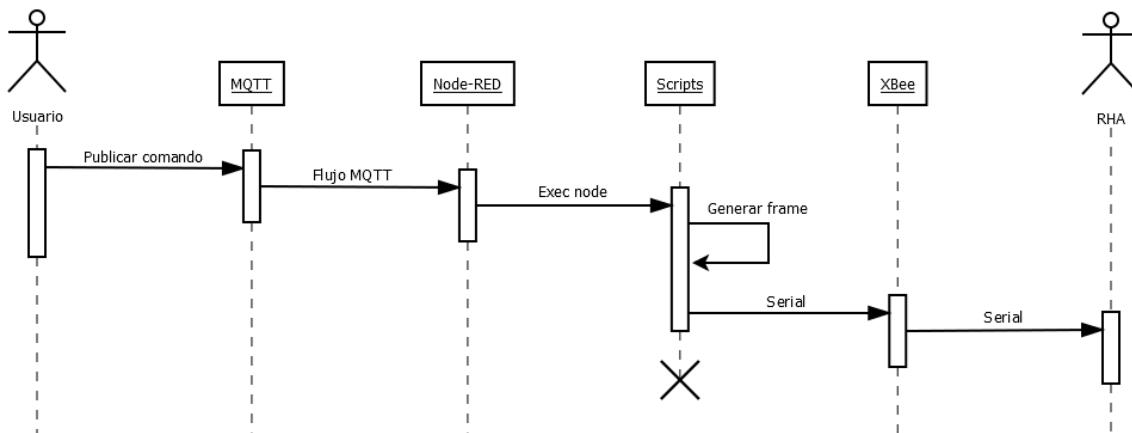


Figura 4.3: Diagrama de Secuencia de envío

estados.

Es posible (y deseable) que ciertos elementos tengan otro control independiente a la red domótica. Así, por ejemplo, una persiana conectada a la red domótica podría ser accionada por el mismo ordenador sin obviar que el mecanismo podría accionarse a través de un pulsador. Esto hace necesaria la monitorización de la mayor parte posible de elementos. Podría darse el caso de que, incluso, la acción de ciertos elementos fuera mecánica en complementación a la acción de naturaleza eléctrica, imposibilitando cualquier integración de estos métodos alternativos de accionamiento en la red domótica.

La unidad de mando debe encargarse de igual manera de la interacción con el usuario, aportando una interfaz.

Los requisitos de un sistema domótico no son especialmente exigentes en cuanto a la capacidad de procesamiento, por lo que características como un tamaño contenido o un bajo coste se valoran positivamente en la elección del ordenador⁰. En este contexto, se hace uso de la popular Raspberry Pi 3 Model B (figura 1.2) para el cometido descrito.

4.2.1. Raspberry Pi 3 Model B

La Raspberry Pi 3 Model B es uno de los más actuales modelos² de la tercera generación de este popular micro-ordenador de bajo coste. Si se miran sus especificaciones, se puede observar que, corriendo a través de una CPU Broadcom BCM2837 de 64 bits a 1.2GHz, posee:

- 1GB de memoria RAM
- Conexión LAN inalámbrica y módulo Bluetooth integrados
- Puerto Ethernet

²Sólo se encuentra la Raspberry Pi 3 Model B+ con una fecha de lanzamiento posterior

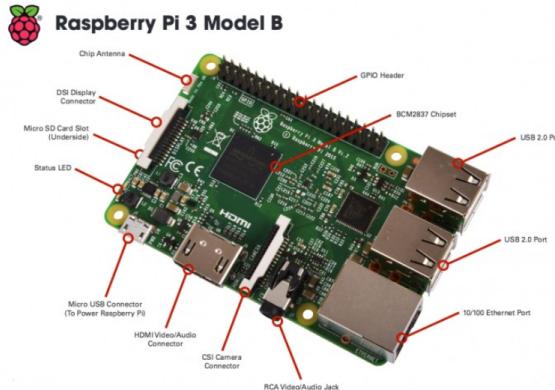


Figura 4.4: Layout de puertos de la Raspberry Pi

- 40 pines de entrada/salida (GPIO)
- 4 puertos USB 2.0
- Salida stereo de 4 polos
- Conector HDMI
- Puerto de cámara CSI
- Puerto de display DSI
- Puerto microSD
- Puerto microUSB

Se puede observar el layout de estos componentes sobre la placa en la figura 4.4.

El puerto microUSB tiene la función de alimentar al ordenador. Se debe conectar una alimentación de 5 voltios a 2.5 amperios. Podría funcionar con una fuente de menos potencia pero podría ser que no soportara la inclusión de ciertos periféricos.

El puerto microSD sirve de alojamiento para la memoria ROM de la Raspberry. A través de este puerto, se carga el sistema operativo y se utiliza la memoria libre como almacenamiento interno.

Raspbian OS

Raspbian OS es el sistema operativo utilizado en la Raspberry Pi del laboratorio (figura 4.5). Se trata de una distribución no oficial basada en Debian adaptada a las especificaciones de la placa computadora Raspberry Pi. Debian está basado en el sistema GNU/Linux y, por tanto, se habla de software libre. El software, con sus características y actualizaciones, es desarrollado por la comunidad.

Entre otras funcionalidades, destaca el poder configurar la Raspberry de manera sencilla a través del menú *raspy-config*

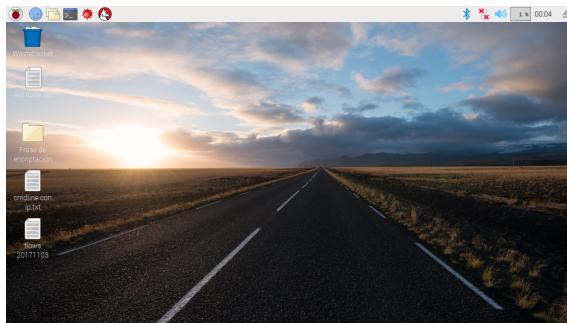


Figura 4.5: Escritorio de Raspbian

En el Anexo A.1 se encuentran las instrucciones para obtener Raspbian 9.1 (Stretch) instalado en la Raspberry. Esta es la última distribución estable a día de hoy.

Comunicación con otros ordenadores

A la hora de trabajar con la Raspberry instalada, no es usual que sea deseable la instalación de periféricos de entrada y salida para interactuar con ella. Es por esto por lo que se recurre a una conexión SSH para trabajar desde remoto con otro ordenador exactamente de igual manera a la que lo haríamos desde la ventana de comandos de Linux.

El procedimiento es diferente en función de si la conexión se efectúa desde una máquina en Linux o en Windows

■ Conexión SSH desde Windows

En Windows existen varios programas dedicados a establecer conexiones entre dispositivos. Uno de los más conocidos es **Putty** (figura 4.6), en cuya interfaz puedes introducir la dirección IP del cliente³, un puerto libre y el tipo de conexión que deseas (SSH, en nuestro caso). Al pulsar *Open* se abre un terminal en el que se solicitan las credenciales antes de tener acceso completo a la Raspberry.

■ Conexión SSH desde Linux

En Linux se puede establecer una conexión SSH haciendo uso del terminal

```
ssh root@190.168.1.104
```

El comando *ssh* establece este tipo de conexiones. El usuario se indica en el lugar de *root* en el ejemplo mientras que la IP del remoto se sitúa después del arroba. Es posible configurar el puerto con la opción *-p* (por defecto se usa el puerto 22).

³La dirección IP de la Raspberry Pi se puede obtener usando el comando *ifconfig* una vez la conexión a internet ya ha sido establecida

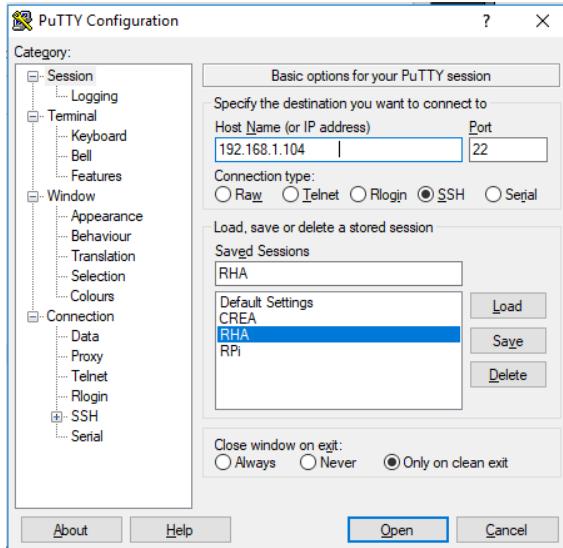


Figura 4.6: Interfaz de Putty

```
ssh -p 22 root@190.168.1.104
```

De igual manera que en Windows, se solicitará la contraseña del usuario si procede y se accederá al terminal.

4.2.2. MQTT

Mosquitto (Message Queue Telemetry Transport) es un protocolo de código abierto enfocado a las conexiones Machine-to-Machine (M2M) [7] que se ha popularizado entre diferentes aplicaciones que precisan de comunicación entre sensores y mandos de redes domóticas. Entre sus características se encuentra un consumo de recursos muy bajo.

Su funcionamiento se basa en una configuración de estrella. Trabaja con un nodo central (también llamado Broker) con el que se establecen conexiones bidireccionales desde múltiples clientes (figura 4.7). Estas conexiones son cifradas, aportando una capa de seguridad a la red domótica.

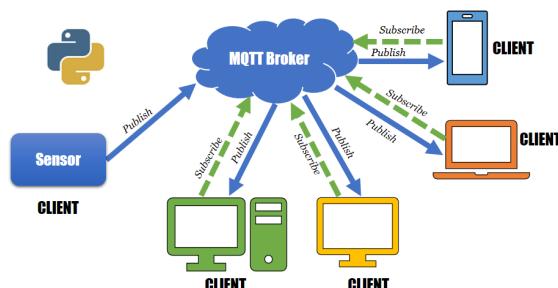


Figura 4.7: Arquitectura MQTT

El concepto de *topic* es la forma que tiene MQTT de articular las comunicaciones. Se puede hacer una analogía de los *topic* con un tablón de anuncios. La gente puede publicar en el tablón lo que le plazca y esto será visto por aquellos que se paren a mirar el tablón. De igual manera funciona MQTT, cualquier cliente puede publicar en un *topic* y este mensaje será recibido por aquellos clientes que estén suscritos a ese mismo *topic*. Los *topics*, además, son jerarquizables. Esto es, pueden generarse subtopics de manera recurrente con el fin de poder enviar mensajes únicamente a un grupo de clientes si se observa la red desde una perspectiva más global.

Existen varios *Broker* de MQTT pero, con diferencia, el más popular es el llamado Mosquitto.

Una guía para su instalación puede encontrarse en el Anexo A.2

MQTT en RoboHealth

En cuanto al funcionamiento dentro de la habitación, existe un topic denominado *Robohealth/room/devices* donde publican los distintos dispositivos mientras Node-RED está suscrito.

Los mensajes en el topic de la habitación siguen un mismo formato:

'id' : xx', 'atrib1' : 'yy', 'atrib2' : 'zz', (...)

xx remplaza el identificador del cliente, único para cada dispositivo. RoboHealth Arm ha sido identificado con el número **99**.

atrib1, *atrib2* y sucesivos son atributos a los que se les quiere dar un valor. En el caso de los dispositivos digitales el atributo suele ser único, siendo denominado *status*. En el caso de RoboHealth Arm es posible trasladarle dos atributos correspondientes a las coordenadas articulares objetivo del robot. Los atributos son **shoulder** para el hombro y **elbow** para el codo.

yy, *zz* y sucesivos son los valores correspondientes a los atributos. En el caso del brazo deberán pasarse las dos **coordenadas articulares en formato hexadecimal**.

4.2.3. Node-RED

Node-RED es una herramienta de programación basada en una interfaz de programación online representada a través de flujos y nodos (figura 4.8). Viene preinstalada en Raspbian Stretch, lo que puede dar una idea del grado de integración que tiene esta herramienta en la comunidad.

Los flujos representan caminos de transmisión de los objetos de node-RED, denominados por defecto *msg*. Estos objetos *msg* poseen unos atributos, algunos creados

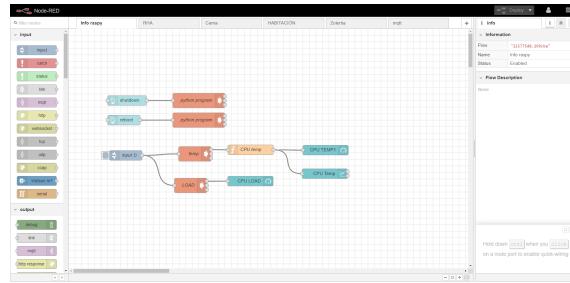


Figura 4.8: Ejemplo de interfaz de Node-RED

por defecto y otros opcionales personalizados. Uno de los atributos por defecto es *payload*, que suele ser utilizado como contenedor de la información a trasladar. Esta información puede ser de casi cualquier tipo, desde un booleano hasta otro objeto con sus propios atributos. El otro de los atributos por defecto es *_msgid*, que es un identificador del mensaje enviado que sirve para monitorizar su estado a lo largo del flujo.

Los nodos son etapas en las que, basándose en diferentes tecnologías, se realiza una acción cuando se produce la entrada del mensaje a través del flujo. Esta acción puede ir desde producir algún tipo de reacción ajena a Node-RED hasta modificar variables internas del programa, modificando (o no) el mensaje de flujo antes de volver a transmitirlo.

Existen nodos relacionados con un gran número de tareas. La comunidad puede aportar sus propios paquetes de nodos, ampliando progresivamente el número de tecnologías compatibles con Node-RED. Algunos de los nodos más representativos y usados en el proyecto Robohealth son los siguientes:

■ Debug node

El nodo *Debug* (figura 4.9) representa una imagen del objeto *msg* o de uno de sus atributos a su paso por un punto concreto del flujo. Normalmente hace uso de la pestaña *debug* de la interfaz de Node-RED.

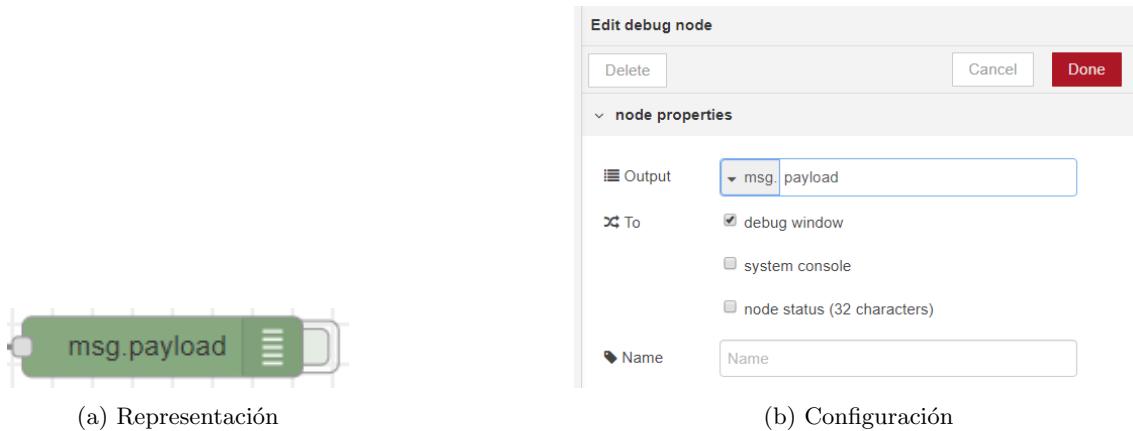


Figura 4.9: Nodo Debug

■ Function node

El nodo *Function* (figura 4.10) contiene una función programada en JavaScript que, por convenio, recibe el objeto *msg* proveniente del flujo y lo utiliza para crear un nuevo objeto (u objetos) *message* que pasar al siguiente nodo del flujo. Existe la posibilidad de no devolver nada si se desea congelar el flujo en ciertas situaciones.



Figura 4.10: Nodo Function

■ Exec node

El nodo *Exec* (figura 4.15) es el nodo de ejecución de comandos. El comando a ejecutar se define en las propiedades, siendo posible añadir el mensaje recibido a través del flujo al final del comando. Las salidas del bloque corresponden a *stdout*, *stderr* y a objetos devueltos. Es posible configurar si la salida debe volcarse al final de la ejecución o en directo durante la ejecución del programa.

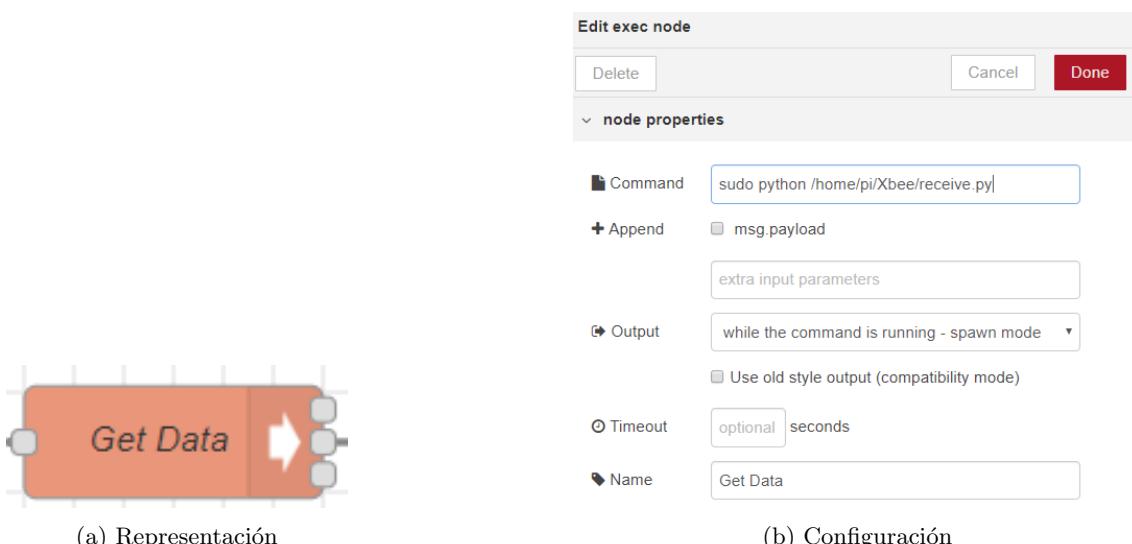


Figura 4.11: Nodo Debug

■ MQTT node

Algunos de los nodos desarrollados por terceros son los relacionados con MQTT. Existen nodos de entrada de MQTT (figura 4.12) y nodos de salida. La configuración disponible incluye la IP, el puerto y el *topic* al que suscribirse o publicar.

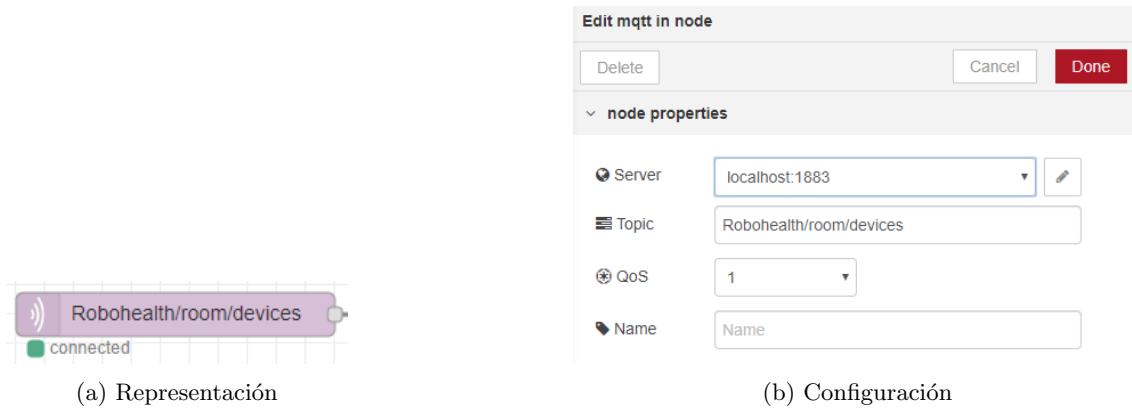


Figura 4.12: Nodo entrada MQTT

■ Nodos de gestión de flujo

Una serie de nodos se usan para modificar el comportamiento normal del flujo de los mensajes *msg*. Posibilitan la aplicación de cierta lógica al programa.

– Inject node

El nodo de inyección introduce un mensaje en el flujo. Se puede programar una inyección periódica, que haga funcionar el flujo de manera similar a un bucle software.

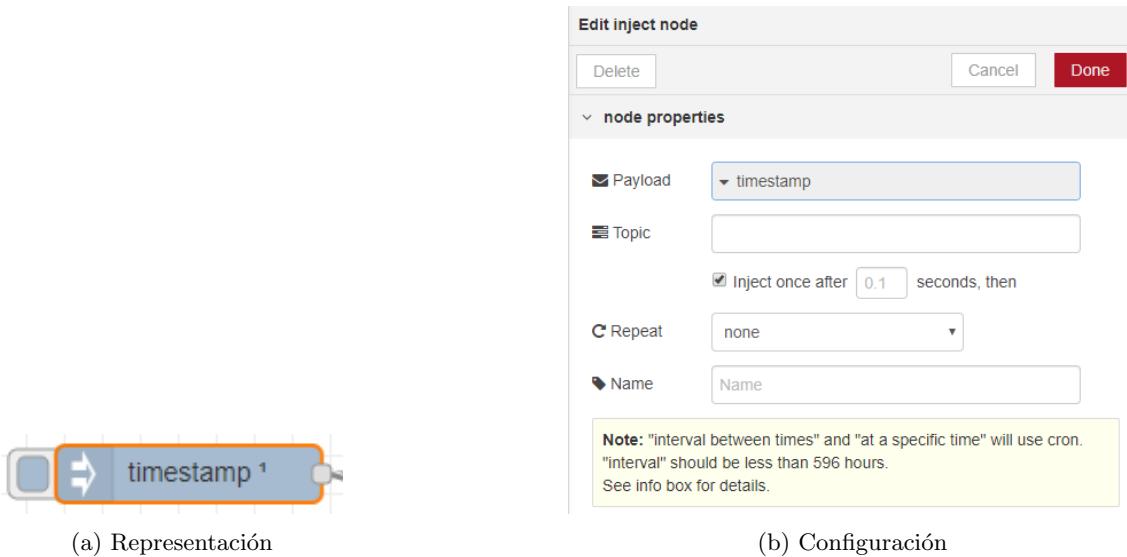


Figura 4.13: Nodo Inject

– Delay node

El nodo *Delay* detiene la ejecución del programa durante el tiempo especificado.

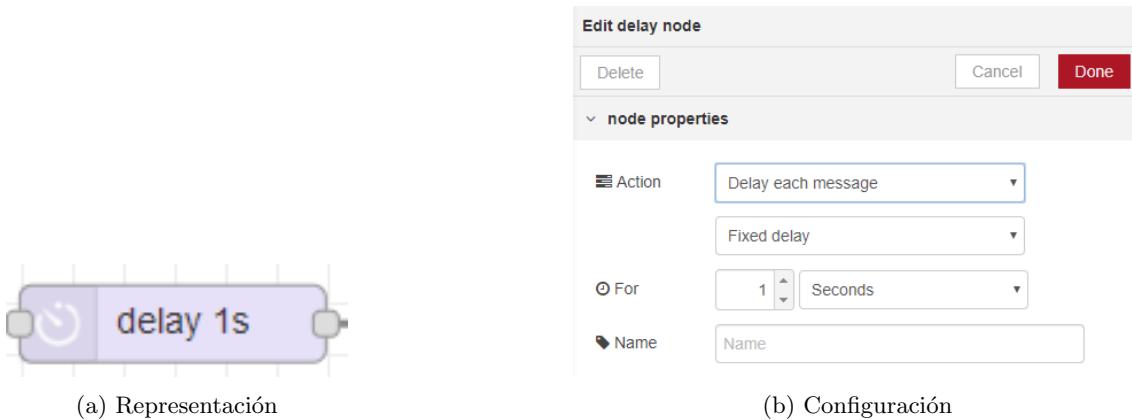


Figura 4.14: Nodo Delay

– Switch node

El nodo *Switch* compara algún atributo del mensaje *msg* con unos valores programados y asignados a cada una de las salidas. Es equivalente a la expresión condicional *switch-case*.

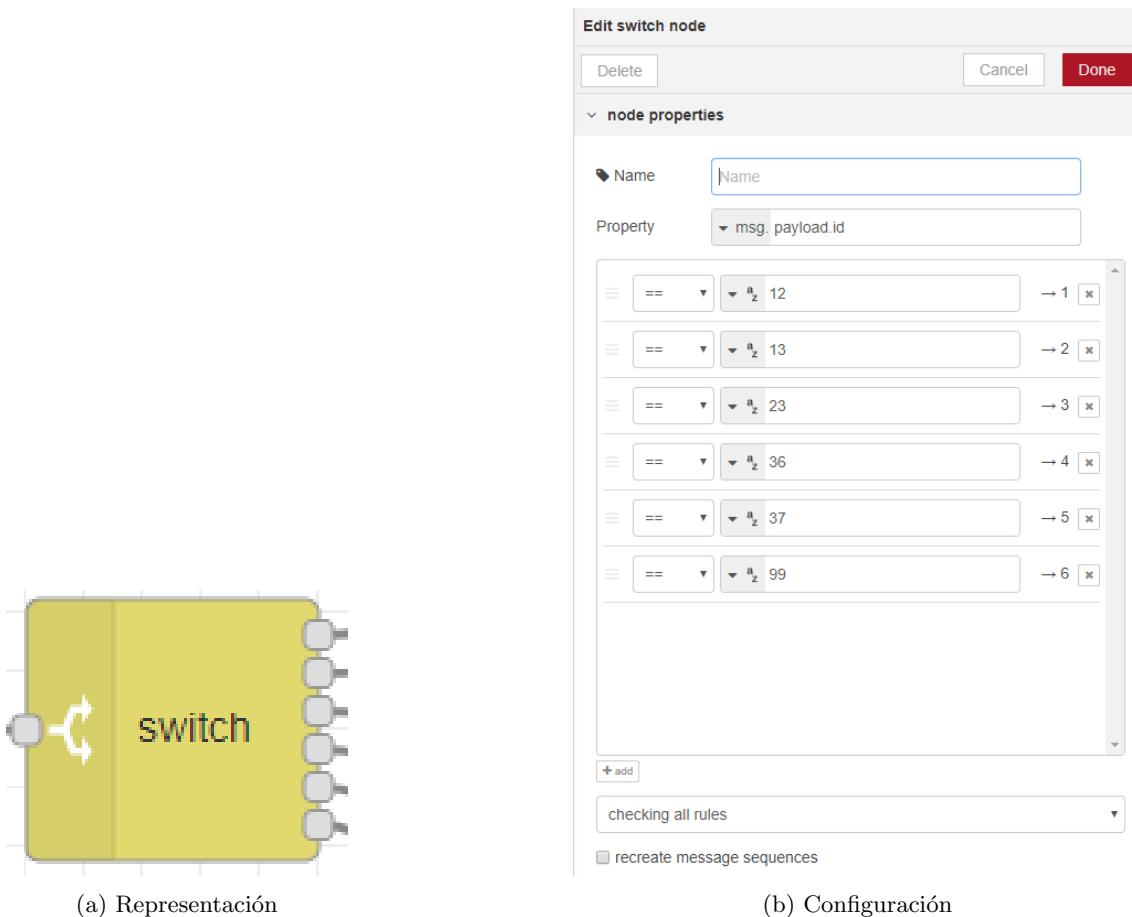


Figura 4.15: Nodo Switch

■ Nodos de UI

Existen varios nodos que representan objetos en el Dashboard de Node-RED

que sirve de interfaz gráfica. Estos nodos pueden ser de entrada o salida de información

– Ejemplos de nodos de entrada

Los nodos de entrada (figura 4.16) permiten introducir mensajes en el sistema. Existen entradas digitales, como pueden ser botones o interruptores, o entradas analógicas, como sliders.



Figura 4.16: Nodos de entrada

– Ejemplos de nodos de salida

Los nodos de salida (figura 4.17) representan el valor de algún atributo del mensaje que les llega. Esta representación puede tener formato de texto, numérico o gráfico.

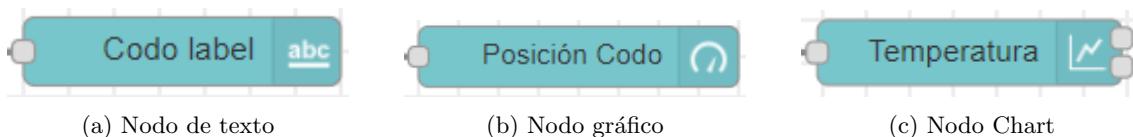


Figura 4.17: Nodos de salida

Node-RED en RoboHealth

La aplicación de Node-RED para los dispositivos de RoboHealth se orienta a la interacción con el usuario a través de una interfaz gráfica. Esta interfaz gráfica se divide en dos layout.

El primer layout (figura 4.18) se encarga del control básico del estado de la Raspberry Pi. Incluye representaciones de la temperatura y el uso de la CPU así como botones para reiniciar y apagar el ordenador.

El flujo (figura 4.19) se basa en la ejecución de comandos para obtener la información de temperatura (1/código 4.2.3) y carga de la CPU (2/código 4.2.3). De igual manera, se lanzan comandos para apagar (3/código 4.2.3) y reiniciar (4/código 4.2.3) la Raspberry Pi. Los comandos utilizados son los siguientes:

```

1 $ vcgencmd measure_temp
2 $ top -d 0.5 -b -n2 | grep "Cpu(s)" | tail -n 1 | awk '{ print $2 "+" $4 }'
3 $ sudo shutdown -h now

```

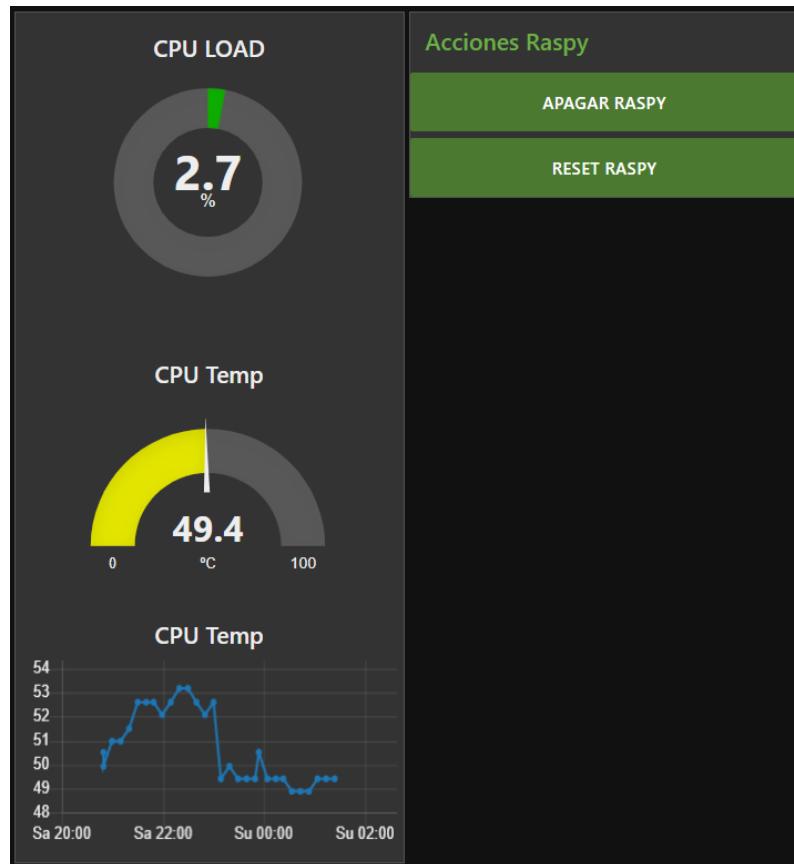


Figura 4.18: UI de control elemental

```
4 | $ sudo reboot
```

El segundo layout incluye el control de los dispositivos previamente desarrollado. El Dashboard (figura 4.20) recoge la UI para controlar cada bloque de dispositivos. A continuación se recogen ejemplos de los flujos correspondientes a la cama (figura 4.21), la habitación (figura 4.22⁴) o los dispositivos Zolertia (figura 4.23).

Node-RED para RoboHealth Arm

Para el control de RoboHealth Arm se ha desarrollado un flujo en Node-RED que se explica a continuación.

En la figura 4.24 se muestra el flujo de encendido del brazo robótico

⁴En la imagen se recoge únicamente el flujo correspondiente a las persianas

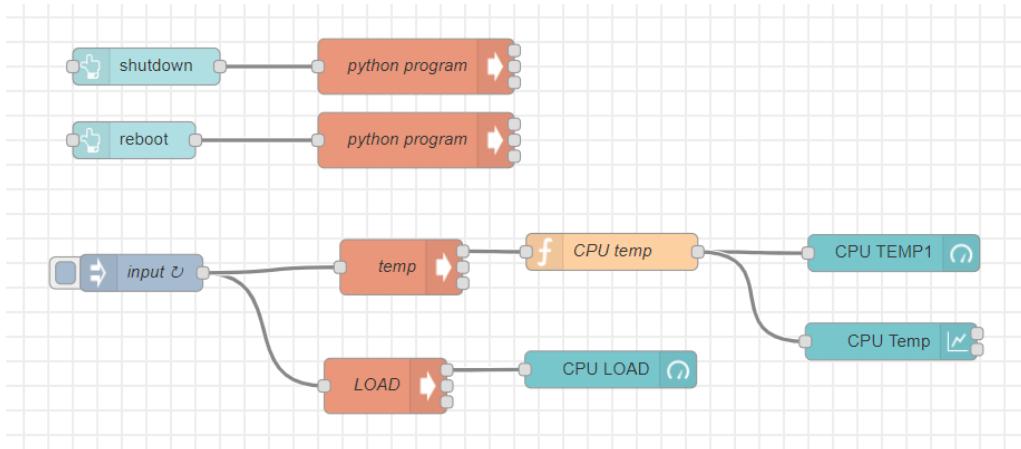


Figura 4.19: Flujo de control elemental



Figura 4.20: Dashboard

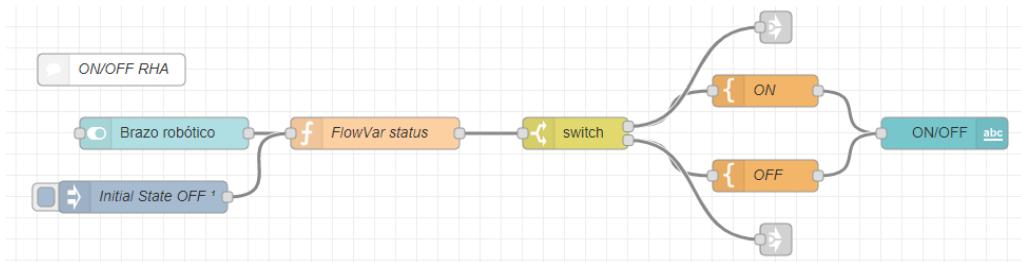


Figura 4.24: Flujo de encendido de RHA

El flujo se basa en la obtención el estado del control del brazo a través del botón ON/OFF y su asignación a una variable de flujo. Esta asignación se efectúa a través de la función *FlowVar status* (código 4.1)

Código 4.1: FlowVar status

```

1 if (msg.payload === true) {
2   flow.set("status", "ON");
3 }
4 else if (msg.payload === false) {
5   flow.set("status", "OFF");
  
```

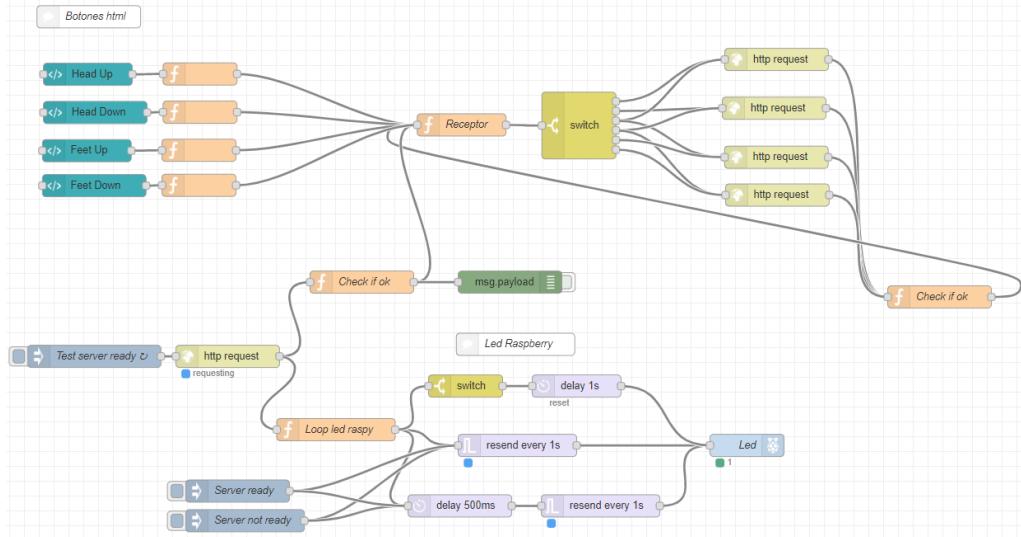


Figura 4.21: Flujo de control de la cama

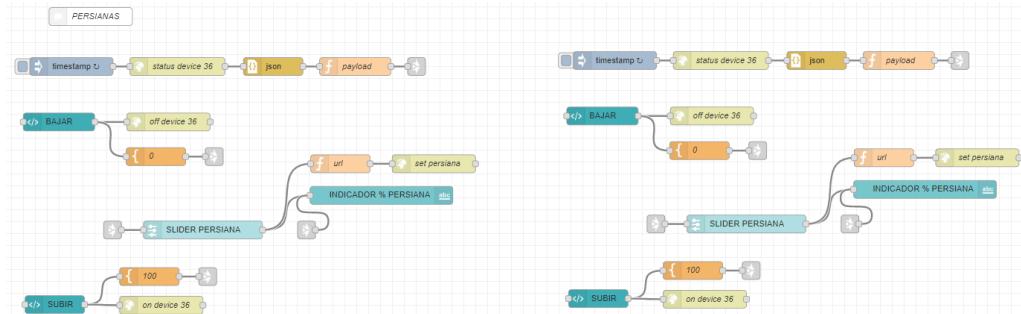


Figura 4.22: Flujo de control de las persianas

```

6 }
7     msg.payload = flow.get("status")
8     return msg;
  
```

Se incluye también en el flujo la inyección del estado inicial OFF al comenzar la ejecución.

El flujo de recepción de la información de RHA se programa como en la figura 4.25

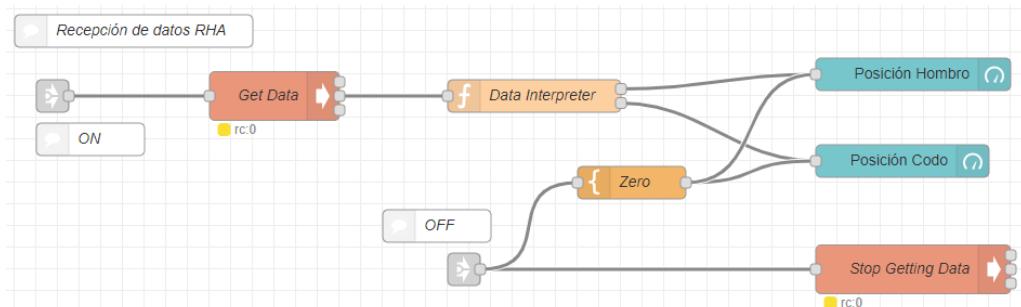


Figura 4.25: Flujo de recepción de RHA

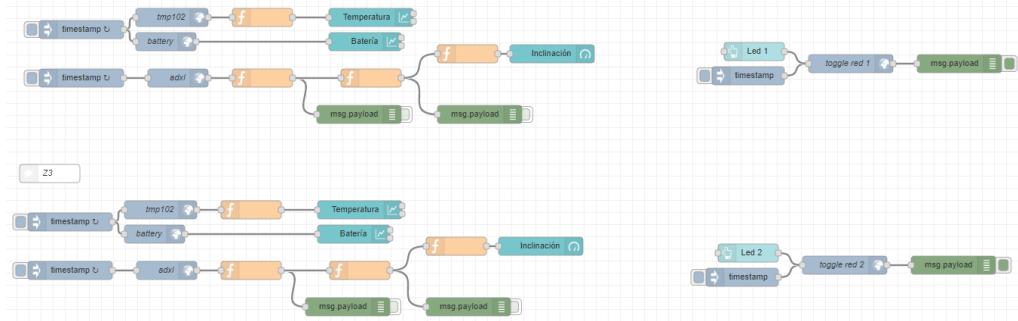


Figura 4.23: Flujo de control de Zolertia

La recepción de un flanco de encendido, pone a correr el comando ?? que lanza un script que devuelve los datos recibidos. Estos datos se interpretan con la función *Data interpreter* (código 4.2) y se muestra la información en el Dashboard con una salida gráfica

```
1 $ sudo python /home/pi/Xbee/receive.py
```

Código 4.2: Data Interpreter

```
1 var num1_str = msg.payload[1]+msg.payload[2]+msg.payload[3];
2 if (msg.payload.length < 11) {
3     var num2_str = msg.payload[6]+msg.payload[7];
4 }
5 else {
6     var num2_str = msg.payload[6]+msg.payload[7]+msg.payload[8];
7 }
8 msg.payload = num1_str;
9 var newMsg = {payload: num2_str}
10 return [msg, newMsg];
```

Un flanco de apagado detiene el script con el comando ?? y manda un 0 a la representación de la información

```
1 $ sudo pkill -f receive.py
```

El flujo de configuración de la emisión de datos se encuentra en la figura 4.26. En resumen, obtiene los datos de posiciones articulares y modo de comunicación de la interfaz gráfica y los asigna a variables de flujo. Las posiciones articulares están limitadas por los slider: para el codo, los valores estan entre **60** y **110**; para el hombro, entre **125** y **166**. Las funciones *FlowVar elbow*, *FlowVar shoulder* y *FlowVar ComMode* son similares a *FlowVar status* (código 4.1). El flujo también establece los valores iniciales para codo, hombro y modo de comunicación; siendo 60, 125 y AT respectivamente.

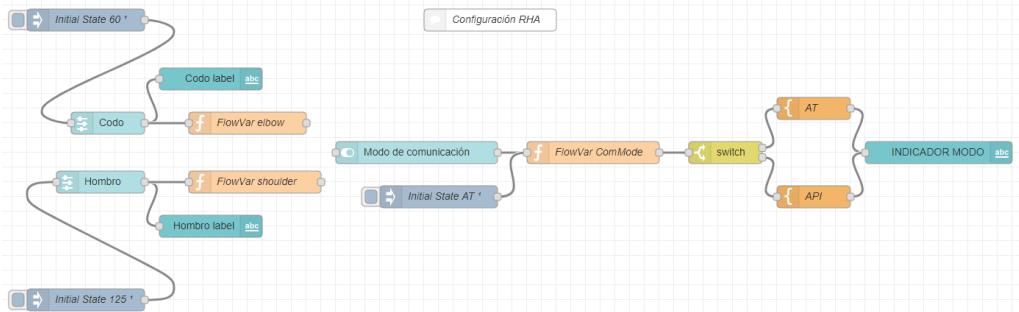


Figura 4.26: Flujo de configuración de RHA

En la figura 4.27 se recoge el flujo correspondiente a la emisión de datos al RHA. Comprueba el tipo de comunicación y elige el script a correr de acuerdo a ello. Previamente, obtiene las coordenadas articulares para pasárselas a los scripts.

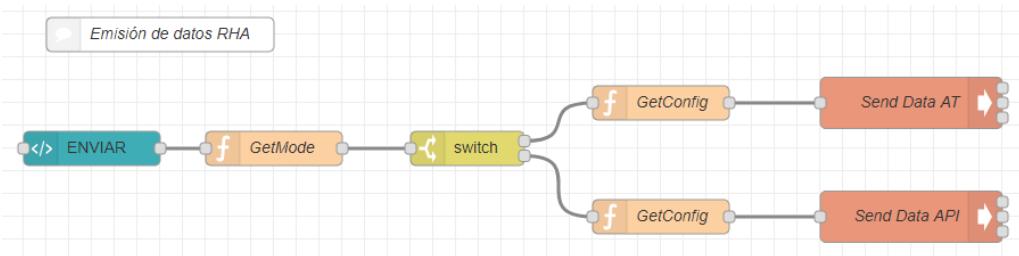


Figura 4.27: Flujo de emisión de RHA

Función *GetMode*:

Código 4.3: GetMode

```

1 if (flow.get("ComMode") === undefined) {
2     flow.set("ComMode", "AT");
3 }
4 if (flow.get("status") === "ON") {
5     msg.payload = flow.get("ComMode");
6 }
7 return msg;

```

Función *GetConfig*:

Código 4.4: GetConfig

```

1 var posX = flow.get("RHA_shoulder");
2 var posY = flow.get("RHA_elbow");
3 var hexPosX = posX.toString(16);
4 var hexPosY = posY.toString(16);
5 msg.payload = [hexPosX, hexPosY];
6 return msg;

```

Comandos para enviar los mensajes AT y API. En la práctica, *msg.payload* se añade al final del comando.

```

1 $ sudo python /home/pi/Xbee/send-AT.py
2 $ sudo python /home/pi/Xbee/send-API.py

```

4.2.4. Python scripts

En la Raspberry Pi estan guardados una serie se scripts en lenguaje Python que son los encargados de enviar (tanto en modo AT, como en modo API) y recibir la información.

Unos comentarios sobre la edición y compilación de scripts escritos en Python en una máquina que corra Raspbian y, en general, cualquier sistema operativo basado en GNU/Linux, pueden ser encontrados en el Anexo A.3.

SendAT.py

SendAPI.py

Receive.py

4.3. Transmisión de la información

4.3.1. XBee Shield

4.3.2. Módulos XBee

Modos de comunicación

XCTU

Perfiles de comunicación

4.4. RoboHealth Arm

4.4.1. Protocolo de comunicación RHA

4.4.2. Modificaciones a RHA

Modificaciones Hardware

Modificaciones Software

4.5. Integración

4.5.1. MQTT - Node-RED

Flujos MQTT

4.5.2. User Interface - Node-RED

Flujos UI/Node-RED

4.5.3. Node-Red - XBee

4.5.4. XBee - RHA

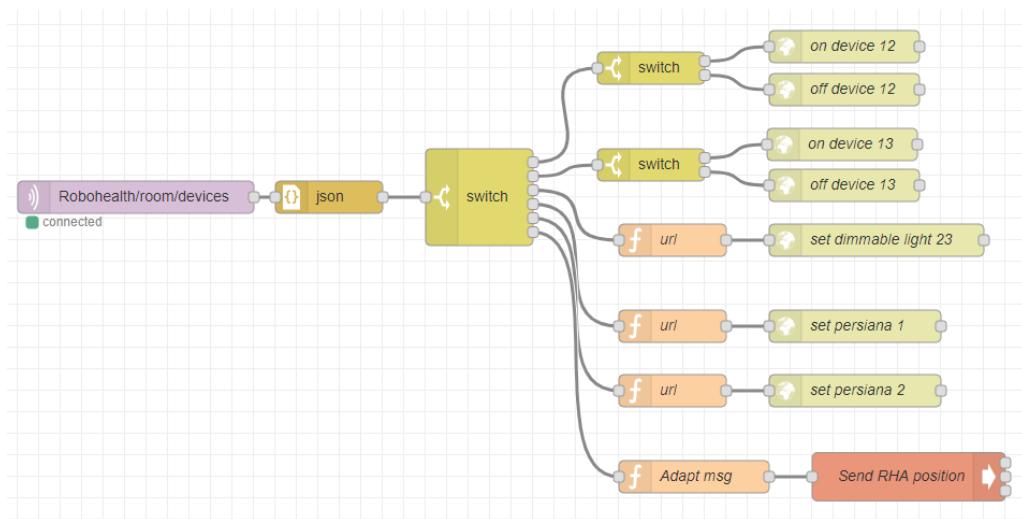


Figura 4.28: Flujo de control de Zolertia

Capítulo 5

Resultados y discusión

En este capítulo...

5.1. Resultados

Los resultados son una parte imprescindible del TFG. Muestran lo que realmente se ha hecho y deben ser explicados con rigor y claridad

5.1.1. Test User Interface

5.1.2. Test MQTT

5.2. Discusión

Una vez expuestos los resultados en la sección anterior, aquí se deben comentar y analizar su validez.

Capítulo 6

Gestión del proyecto

En este capítulo se describe la gestión del proyecto: ciclo de vida, planificación, presupuesto, etc.

6.1. Ciclo de vida

Explicación de las fases del proyecto: definición, análisis, diseño, construcción, pruebas, implementación, validación, documentación. Ejemplo: diagrama de Pert.

6.2. Planificación

Se puede indicar mediante un diagrama de Gantt.

6.2.1. Planificación inicial

6.2.2. Planificación final

6.3. Presupuesto

6.3.1. Personal

6.3.2. Material

6.3.3. Resumen de costes

Capítulo 7

Conclusiones

Se presentan a continuación las conclusiones...

7.1. Conclusión

Una vez finalizado el proyecto...

7.2. Desarrollos futuros

Un posible desarrollo...

Apéndice A

Anexo A

El Anexo A recoge y explica los procesos de instalación y configuración de los diferentes sistemas operativos, programas y herramientas necesarios para la implementación del proyecto.

A.1. Instalación de Raspbian OS

Raspbian es un sistema operativo orientado a Raspberry Pi por lo que existe multitud de documentación complementaria en la web oficial[8]

Antes de comenzar a instalar el sistema operativo en una tarjeta microSD, se debe comprobar que ésta reúne los requisitos para ser usada en esta aplicación. Uno de estos requisitos es que la capacidad de la tarjeta sea superior a 8GB. Sin embargo, el uso de tarjetas de un tamaño superior a 32GB hace que sea necesario formatear la tarjeta antes de instalar el sistema operativo. Esto es debido a que el formato de serie (exFAT) no es compatible con el bootloader de Raspberry Pi, por lo que se deberá aplicar el formato FAT16 o FAT32 previamente.

Las instrucciones están pensadas teniendo en cuenta que se posee un ordenador con sistema operativo Windows.

1. Descargar el sistema operativo

Se puede descargar la imagen desde la web de descargas de Raspberry Pi¹

2. Descargar e instalar Etcher

Se precisa de una herramienta de escritura de imágenes y Etcher es la solución más sencilla para la mayoría de usuarios. Permite la escritura de la imagen sin la necesidad de extraer el archivo zip.

3. Escribir la imagen a una microSD

¹<https://www.raspberrypi.org/downloads/raspbian/>



Figura A.1: Descarga de Raspbian Stretch

```
pi@raspberrypi:~ $ lsb_release -a
No LSB modules are available.
Distributor ID: Raspbian
Description:    Raspbian GNU/Linux 9.1 (stretch)
Release:        9.1
Codename:      stretch
pi@raspberrypi:~ $
```

Figura A.2: Versión de Raspbian

Seleccionar el archivo de la imagen la SD de destino es suficiente para flashear la imagen.

4. Una vez acabado el proceso de descompresión, la instalación debería estar completa. Introduciendo la tarjeta micro SD en su correspondiente posición en la Raspberry Pi, ésta accederá a Raspbian para arrancar el sistema.
5. Para comprobar la versión del SO instalado, se puede hacer uso del comando *lsb-release*

```
1  lsb_release -a
```

A.2. Instalación MQTT en Raspbian OS

Mosquitto (broker de MQTT) se instala de igual manera que cualquier otra aplicación, haciendo uso de los repositorios.

En primer lugar, se actualizan los repositorios.

```
1  sudo apt-get update
```

Posteriormente, se instala Mosquitto.

```
1  sudo apt-get install mosquitto
```

Si todo va bien, al terminar el proceso, MQTT debería estar instalado.

Para probar la instalación, se pueden usar clientes de MQTT para enviar y recibir información a través un topic de prueba. Estos clientes son, por ejemplo, *IoT MQTT Dashboard* en Android o *MQTT.fx* en Windows.

A.3. Edición y compilación de scripts en Raspbian OS

Apéndice B

Anexo B

En el Anexo B se sitúan los desarrollos software íntegros que forman parte del proyecto.

B.1. Código SendAT.py

Código B.1: SendAT.py

```
1 #!/usr/bin/env python
2 import sys
3 import time
4 import serial
5 import logging
6
7 logging.basicConfig(filename='/home/pi/Xbee/SendLogs/sentAT.log', level=logging
8 .DEBUG, format='%(asctime)s - %(message)s')
9
10 ser = serial.Serial(
11     port='/dev/ttyACM0',
12     baudrate = 115200,
13     parity=serial.PARITY_NONE,
14     stopbits=serial.STOPBITS_ONE,
15     bytesize=serial.EIGHTBITS,
16     timeout=1
17 )
18
19 if len(sys.argv)<2:
20     param = bytearray([0x8C,0x5A])
21     logging.warning('AT_Package_non-definde_values')
22 else:
23     param = sys.argv[1]
24
25 val1 = param[0]+param[1]
26 val2 = param[3]+param[4]
27 var1 = int(val1,16)
28 var2 = int(val2,16)
29 checksum_rha = (~(6+2+var1+var2+125) & 0xFF)
30
31 if var1>124 and var1<167 and var2>59 and var2<111:
32     values = bytearray([0xFF, 0xFF, 0x06, 0x02, 0x7D, var1, var2,
33                         checksum_rha])
34     ser.write(values)
35     logging.info('AT_Package_sent')
36 else:
37     logging.warning('AT_Package_out_of_range')
```

B.2. Código SendAPI.py

Código B.2: SendAPI.py

```

1 #!/usr/bin/env python
2 import sys
3 import time
4 import serial
5 import logging
6
7 logging.basicConfig(filename='/home/pi/Xbee/SendLogs/sentAPI.log', level=
8 logging.DEBUG, format='%(asctime)s - %(message)s')
9
10 ser = serial.Serial(
11     port='/dev/ttyACM0',
12     baudrate = 115200,
13     parity=serial.PARITY_NONE,
14     stopbits=serial.STOPBITS_ONE,
15     bytesize=serial.EIGHTBITS,
16     timeout=1
17 )
18
19 if len(sys.argv)<2:
20     param = bytearray([0x8C,0x5A])
21     logging.warning('API_Package_non-defined_values')
22 else:
23     param = sys.argv[1]
24
25 val1 = param[0]+param[1]
26 val2 = param[3]+param[4]
27 var1 = int(val1,16)
28 var2 = int(val2,16)
29
30 checksum_rha = (~(6+2+var1+var2+125) & 0xFF)
31 checksum_xbee = (~(16+1+5*255+254+6+2+var1+var2+125+checksum_rha) & 0xFF)
32
33 if var1>124 and var1<167 and var2>59 and var2<111:
34     values = bytearray([0x7E, 0x00, 0x16, 0x10, 0x01, 0x00, 0x00, 0x00, 0
35         x00, 0x00, 0x00, 0xFF, 0xFF, 0xFE, 0x00, 0x00, 0xFF, 0xFF, 0
36         x06, 0x02, 0x7D, var1, var2, checksum_rha, checksum_xbee])
37
38     #print(hex(values[x]))
39
40     ser.write(values)
41     logging.info('API_Package_sent')
42
43 else:
44     logging.warning('API_Package_out_of_range')

```

B.3. Código Receive.py

Código B.3: Receive.py

```

1 #!/usr/bin/env python
2 import sys
3 import time
4 import serial
5 import struct
6 import logging
7
8 logging.basicConfig(filename='/home/pi/Xbee/ReceiveLogs/receiveAT.log', level=
9 logging.DEBUG, format='%(asctime)s - %(message)s')
10
11 ser = serial.Serial(
12     port='/dev/ttyACM0',
13     baudrate = 115200,
14     parity=serial.PARITY_NONE,
15

```

```
14         stopbits=serial.STOPBITS_ONE,
15         bytesize=serial.EIGHTBITS,
16         timeout=1
17     )
18
19     def serialReader():
20         b = ser.read()
21         if b:
22             bs = struct.unpack("<B",b)
23             return int(bs[0])
24         else:
25             return -1
26
27     while True:
28         if serialReader() == 255 and serialReader() == 255:
29             length = serialReader()
30             data = [0]*length
31             for x in xrange(length):
32                 data[x] = serialReader()
33             if data[0] == 0: # 0 means an UPDATE_INFO package
34                 dev = [data[6],data[11]]
35                 logging.info(data)
36                 print(dev, file=sys.stderr)
```


Apéndice C

Anexo C

El Anexo C recoge la documentación de interés de distintos componentes del proyecto

C.1. Datasheet XBee Shield

Bibliografía

- [1] Ardusmarthome url: <http://openaccess.uoc.edu/webapps/o2/bitstream/10609/64347/8/msanchezmunozOTFM0617memoria.pdf>.
- [2] Controlador central para un sistema domótico utilizando el protocolo inalámbrico zigbee url: <https://riull.ull.es/xmlui/bitstream/handle/915/3085/Controlador%20central%20para%20un%20sistema%20domotico%20utilizando%20el%20protocolo%20inalambrico%20ZigBee.pdf?sequence=1&isAllowed=y>.
- [3] Creando un servidor raspberry pi – xbee en python y conectando un cliente arduino – xbee url: <https://www.internetdelascosas.cl/2015/05/31/creando-un-servidor-raspberry-pi-xbee-en-python-y-conectando-un-cliente-arduino/>
- [4] Dispositivos lógicos programables - la comunicación serie. url: http://perso.wanadoo.es/pictob/comserie.htm#conexion_de_un_microcontrolador_al_puerto_serie_del_pc.
- [5] Home automation system url: <https://www.digi.com/resources/project-gallery/home-automation-system>.
- [6] Node-red based custom full-room wake-up light url: <https://www.wouterbulten.nl/blog/tech/custom-wake-up-light-with-node-red/>.
- [7] Primeros pasos con mqtt url: <https://ricveal.com/blog/primeros-pasos-mqtt/>.
- [8] Raspberry pi official website url: <https://www.raspberrypi.org/documentation/>.
- [9] Raspberry pi3 + xbee + xbmq + mqtt + node-red iot url: <https://www.instructables.com/id/Raspberry-Pi3-XBee-MQTT-Node-Red-IoT/>.
- [10] Room wake-up light: Custom room-wide wake-up light using home assistant url: <https://www.wouterbulten.nl/blog/tech/custom-wake-up-light-with-home-assistant/>.
- [11] Sistemas de radiocomunicaciones móviles. url: <https://personal.us.es/murillo/docente/radio/documentos/tema9.pdf>.
- [12] Smart porch light project: Digi's xbee lte cellular module and arduino mega-2560 shine together url: <https://eu.mouser.com/applications/smart-porch-light-with-digi-lte-xbee/>.

- [13] Solar vehicle url: <https://www.digi.com/resources/project-gallery/university-of-minnesota-solar-vehicle> web oficial: <https://umnsvp.org/>.
- [14] Real Academia de Ingeniería de España. *Diccionario Español de Ingeniería*. 1 edition, 2014.
- [15] National Instruments. *LabVIEW Graphical Programming Course*. National Instruments, Elizabeth Gregory, Malan Shiralkar, Harika Basana, 4.6 edition, 2004.
- [16] Cisco Systems. *Fundamentos de Redes Inalámbricas*. Pearson Educación, S.A., 1 edition, 2006.
- [17] Diego M. Zigootto. *Las mil y una curiosidades de Buenos Aires*. Grupo Norma, 2008.