



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. Nunca nadie me enseñó mejor el valor del trabajo, el esfuerzo y el conocimiento.

A mis padres y a mi hermana.

A Irene.

A las personas que me han acompañado estos años en la escuela y al equipo UPM-Motostudent por hacer estos años diferentes.

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 [13]	19
3.2. Servidor Raspberry Pi-XBee en Python [3]	20
3.3. Smart Porch Light Project [16]	21
3.4. Home Automation System [8]	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 [10]	23
3.7. ArduSmartHome [1]	24
3.8. University of Minnesota – Solar Vehicle [17]	25
4. Diseño 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	34
4.3.	Transmisión de la información	46
4.3.1.	Módulos XBee	46
4.3.2.	XBee Shield	50
4.4.	RoboHealth Arm	52
4.4.1.	Protocolo de comunicación RHA	52
5.	Implementación del proyecto	55
5.1.	Scripts en la Raspberry Pi	55
5.1.1.	SendAT.py	55
5.1.2.	SendAPI.py	56
5.1.3.	Receive.py	56
5.2.	Configuración de módulos XBee	57
5.2.1.	Perfiles de comunicación	61
5.3.	Modificaciones a RHA	62
5.3.1.	Modificaciones Hardware	62
5.3.2.	Modificaciones Software	63
5.4.	Integración MQTT - Node-RED	64
5.4.1.	Flujos MQTT	65
5.5.	Integración Node-Red - XBee	65
5.6.	Integración XBee - RHA	67
6.	Resultados y discusión	69
6.1.	Resultados	69
6.1.1.	Test de eficiencia en recepción	69
6.1.2.	Test de eficiencia en emisión AT	70
6.1.3.	Test de eficiencia en emisión API	71
6.1.4.	Test de tiempos de respuesta	71
6.1.5.	Test de MQTT	72
6.2.	Discusión	73
7.	Gestión del proyecto	75
7.1.	Ciclo de vida	75
7.2.	Planificación	75
7.2.1.	Planificación inicial	75
7.2.2.	Planificación final	75
7.3.	Presupuesto	75
7.3.1.	Personal	75
7.3.2.	Material	75
7.3.3.	Resumen de costes	75
8.	Conclusiones	77
8.1.	Conclusión	77
8.2.	Desarrollos futuros	77
A.	Anexo A	79
A.1.	Instalación de Raspbian OS	79

<i>ÍNDICE GENERAL</i>	xv
A.2. Instalación MQTT en Raspbian OS	80
A.3. Edición y compilación de scripts en Raspbian OS	81
B. Anexo B	83
B.1. Código SendAT.py	83
B.2. Código SendAPI.py	84
B.3. Código Receive.py	84
C. Anexo C	87
C.1. Documentación de XBee Shield	87
C.2. Pruebas del proyecto	88
Bibliografia	91

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	33
4.8. Ejemplo de interfaz de Node-RED	34
4.9. Nodo Debug	35
4.10. Nodo Function	35

4.11. Nodo Debug	36
4.12. Nodo entrada MQTT	36
4.13. Nodo Inject	37
4.14. Nodo Delay	37
4.15. Nodo Switch	38
4.16. Nodos de entrada	38
4.17. Nodos de salida	39
4.18. UI de control elemental	39
4.19. Flujo de control elemental	40
4.20. Dashboard	40
4.21. Flujo de control de la cama	41
4.22. Flujo de control de las persianas	41
4.23. Flujo de control de Zolertia	41
4.24. Flujo de encendido de RHA	42
4.25. Flujo de recepción de RHA	42
4.26. Flujo de configuración de RHA	43
4.27. Flujo de emisión de RHA	44
4.28. Dashboard para el control de RHA	45
4.29. Layout del XBee S2	47
4.30. Pinout del XBee S2	48
4.32. API Frames de recepción	49
4.31. API Frames de transmisión	49
4.33. Ejemplo de red XBee	50
4.34. RoboHealth Arm	52
5.1. Interfaz de XCTU	57
5.2. Carga de firmware en XCTU	58
5.3. Propiedades del XBee Coordinador AT	59
5.4. Propiedades del XBee Router AT	60
5.5. Interfaz de consola de XCTU	61
5.6. Flujo de suscripción a MQTT	65
5.7. Flujo de publicación en MQTT	66
5.8. Conexión serial XBee-RPi	67
5.9. Conexión serial alternativa XBee-RPi	67
5.10. Conexión serial Arduino Mega - XBee	68
6.1. Test de recepción de datos	70
6.2. Test de emisión AT de datos	71
6.3. Detalle de la respuesta	71
6.4. Test de uso vía MQTT	72
A.1. Descarga de Raspbian Stretch	80
A.2. Versión de Raspbian	80
C.1. Esquemático de XBee Shield	87
C.2. Test de recepción de datos AT	88
C.3. Test de recepción de datos genérico 1	88
C.4. Test de recepción de datos genérico 2	88
C.5. Test de recepción de datos MQTT	88

C.6. Test de emisión AT de datos	89
C.7. Detalle de la respuesta ante un paquete de mando	89
C.8. Test de uso MQTT	90

Índice de tablas

4.1.	Alternativas de radiofrecuencia	46
4.2.	Estructura de un API Frame	48
4.3.	Estructura de un paquete genérico de comunicación RHA	53
4.4.	Estructura del frame de comando[5]	53
4.5.	Estructura del frame de estado[5]	54
C.1.	Estructura del frame de comando	89

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-

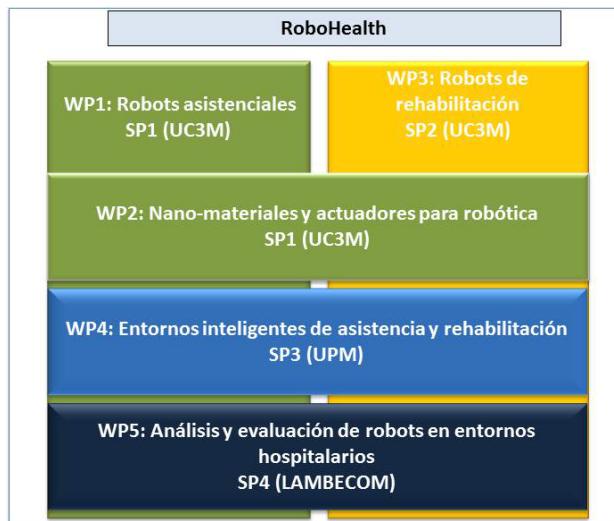


Figura 1.1: Estructura RoboHealth

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.

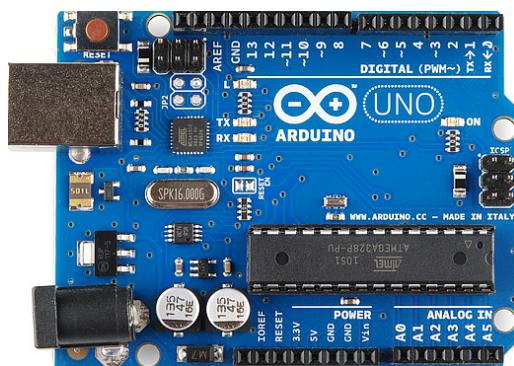


Figura 1.3: Arduino Uno R3

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

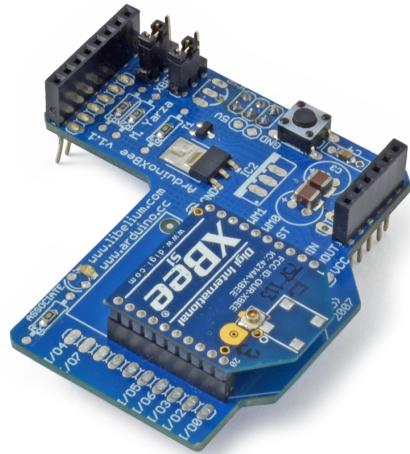


Figura 1.4: Xbee Shield

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

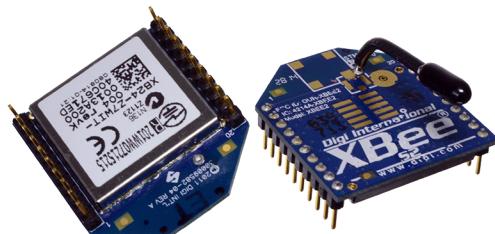


Figura 1.5: Xbee Module

- **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.

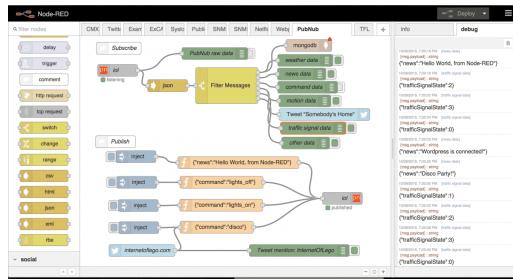


Figura 1.7: Interfaz de edición de Node-RED

- **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 [6]; 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.

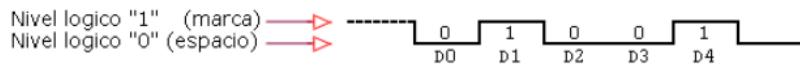


Figura 2.1: Formato serie marca/espacio

Existen varias configuraciones de las líneas de comunicación [6], 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).

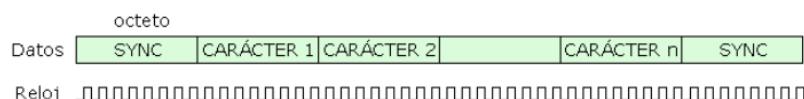


Figura 2.2: Transmisión serial síncrona

- 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

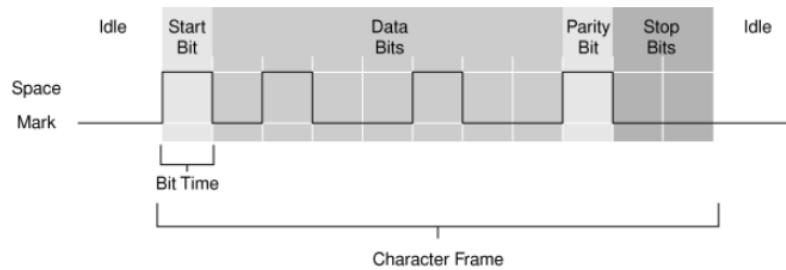


Figura 2.3: Transmisión serial asíncrona

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 [22], 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 [23].

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 [21] (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-

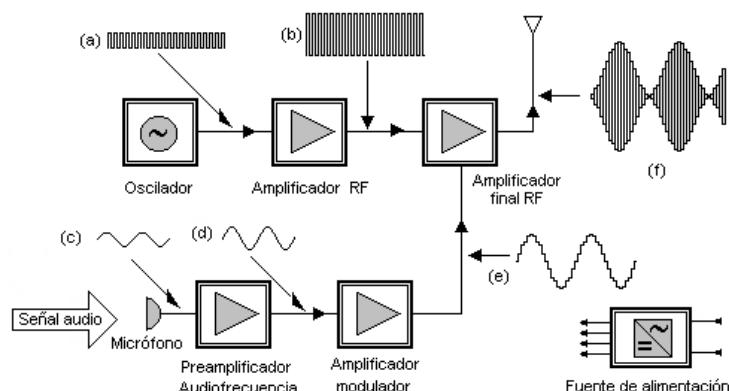


Figura 2.4: Ejemplo de radiotransmisor AM

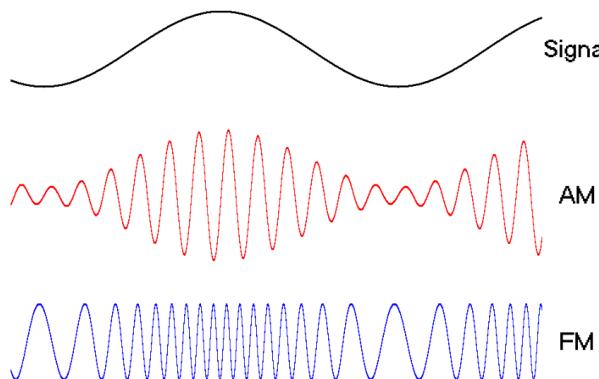


Figura 2.5: Modulación de la señal

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 [21], o QAM.
- Modulación de doble banda lateral [21], o DSB.
- Modulación de banda lateral única [21], 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,

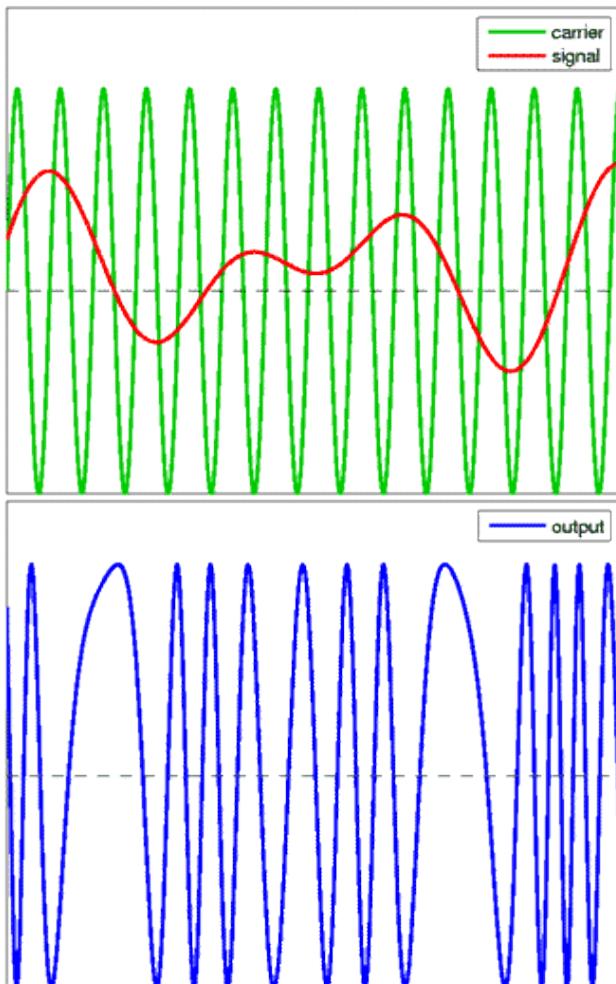


Figura 2.6: Modulación de fase

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 [15]. De forma análoga, uno se puede encontrar con las siguientes configuraciones:

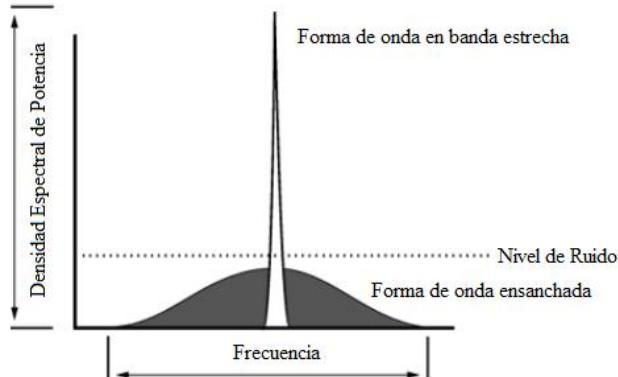


Figura 2.7: Banda estrecha vs DSSS

- 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.

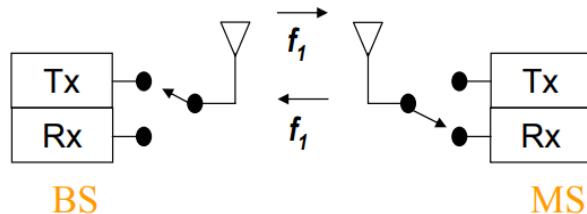


Figura 2.8: Radiocomunicación simplex a una frecuencia

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

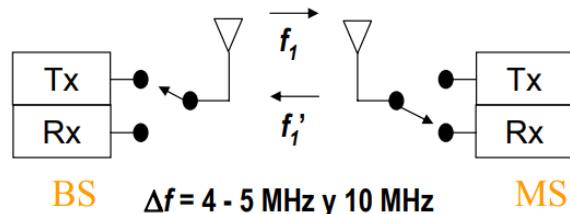


Figura 2.9: Radiocomunicación simplex a dos frecuencias

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

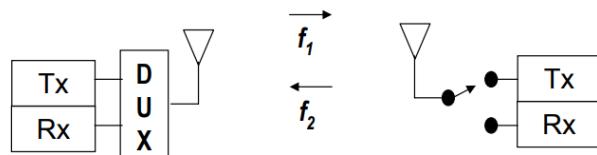


Figura 2.10: Radiocomunicación semiduplex

- 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

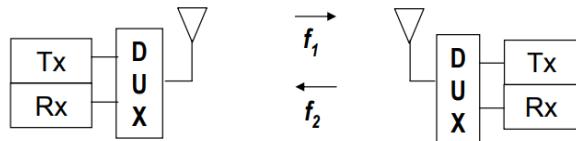


Figura 2.11: Radiocomunicación duplex

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 [23].

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 [24]. 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 [13]

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

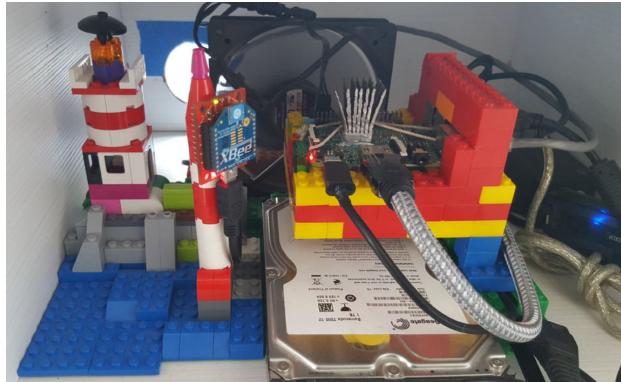


Figura 3.1: Montaje del proyecto 3.1

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 Sheeet 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.

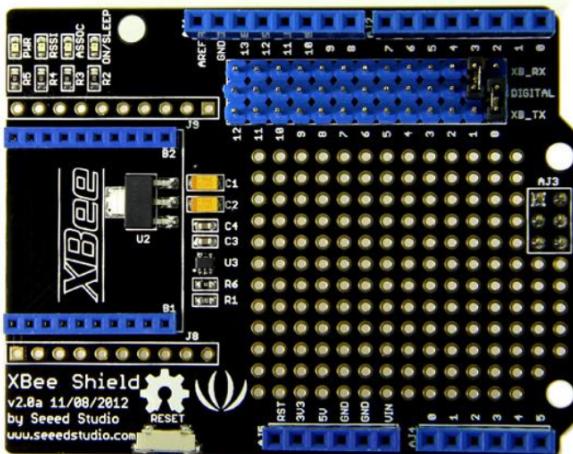


Figura 3.2: XBee Shield v2.0 - Seeed Studio

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 [16]

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.

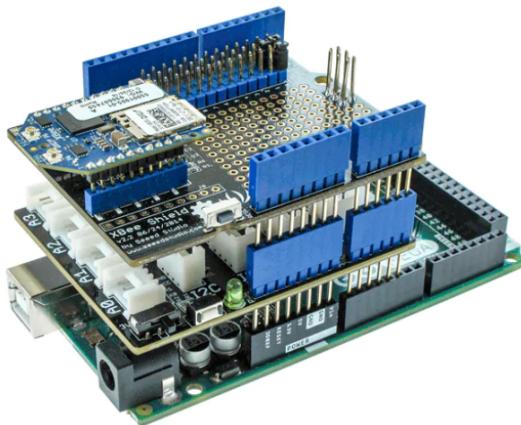


Figura 3.3: Stack Hardware del proyecto 3.4

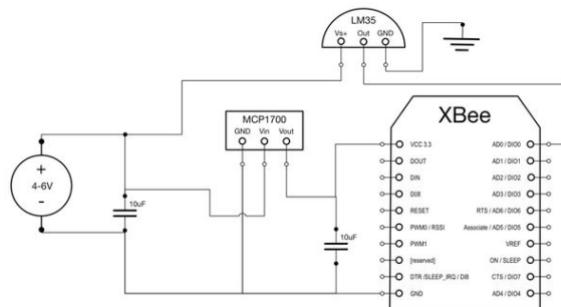


Figura 3.4: Esquema de conexión del sensor de temperatura

3.4. Home Automation System [8]

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

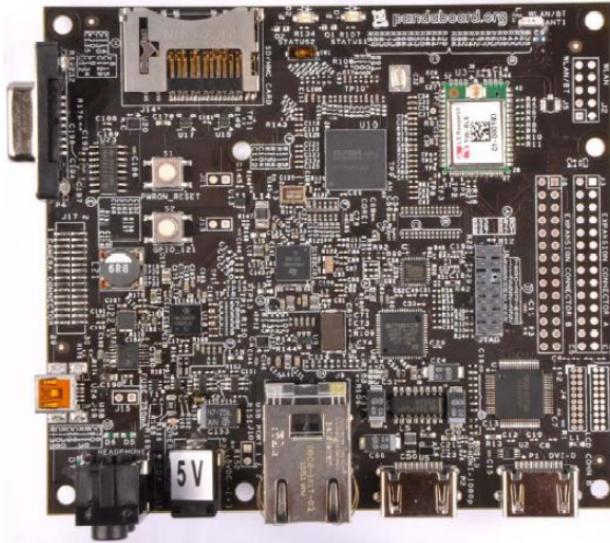


Figura 3.5: PandaBoard ES

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 mas 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 [10]

Proyecto que desarrolla una aplicación Node-RED con el fin de configurar un patrón despertador configurable. El desarrollo del proyecto referenciado [10] se centra en la definición de los flujos de Node-RED puesto que el sistema domótico se ha desarrollado previamente [14].

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

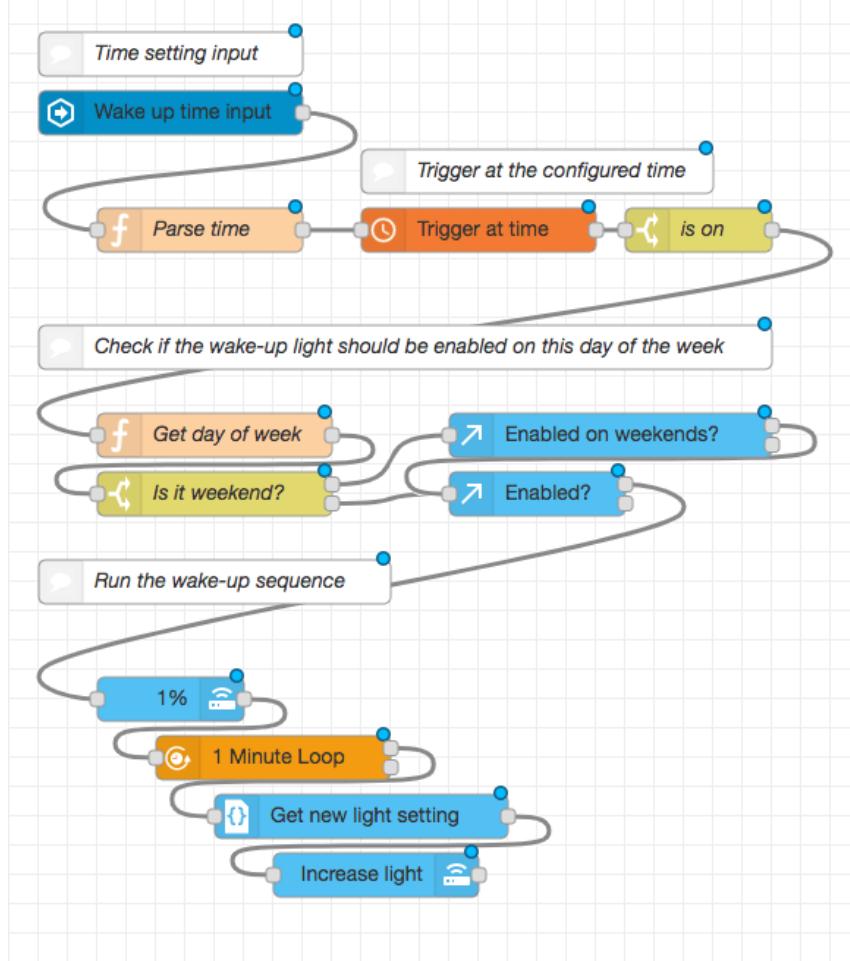


Figura 3.6: Flujo de Node-RED

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

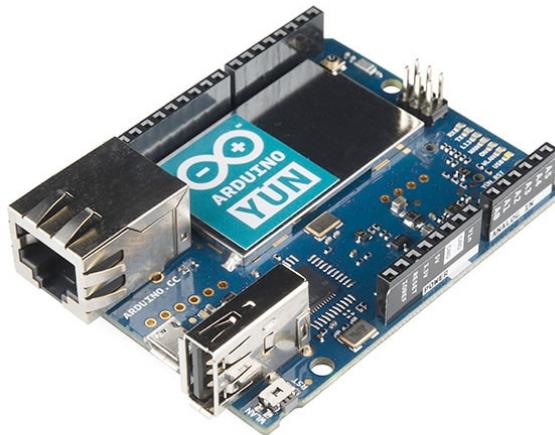


Figura 3.7: Arduino Yun

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 [17]

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

Diseño 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.

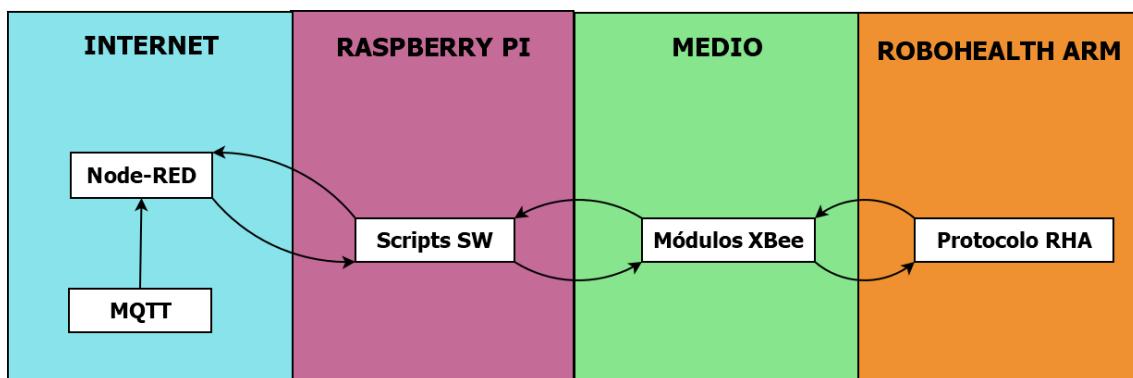


Figura 4.1: Diagrama de interacción

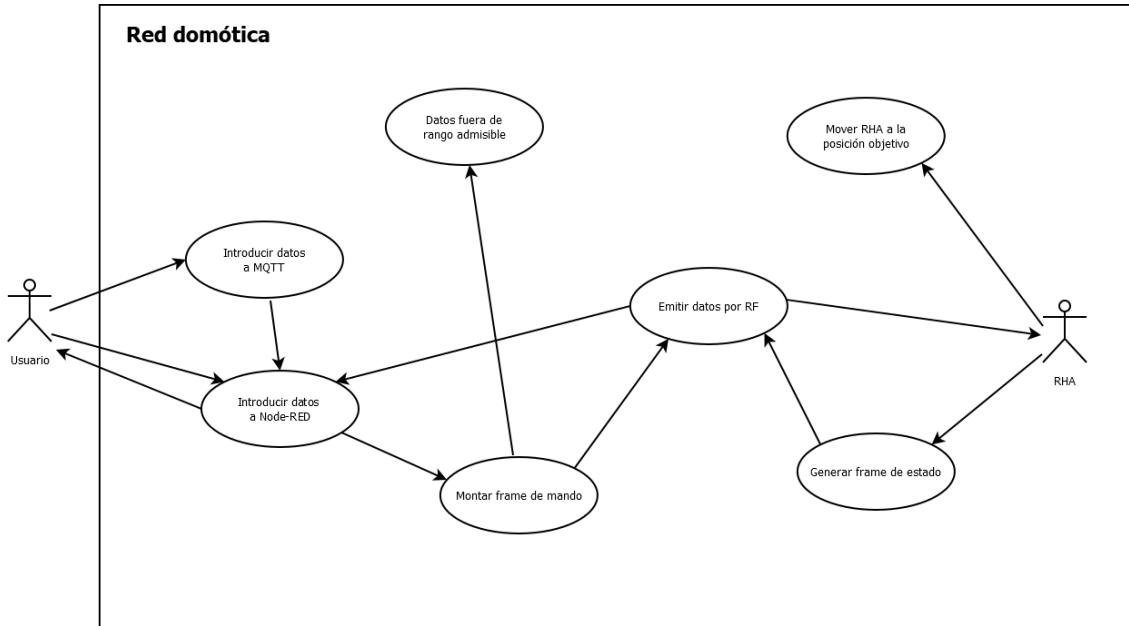


Figura 4.2: Diagrama de casos de uso

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

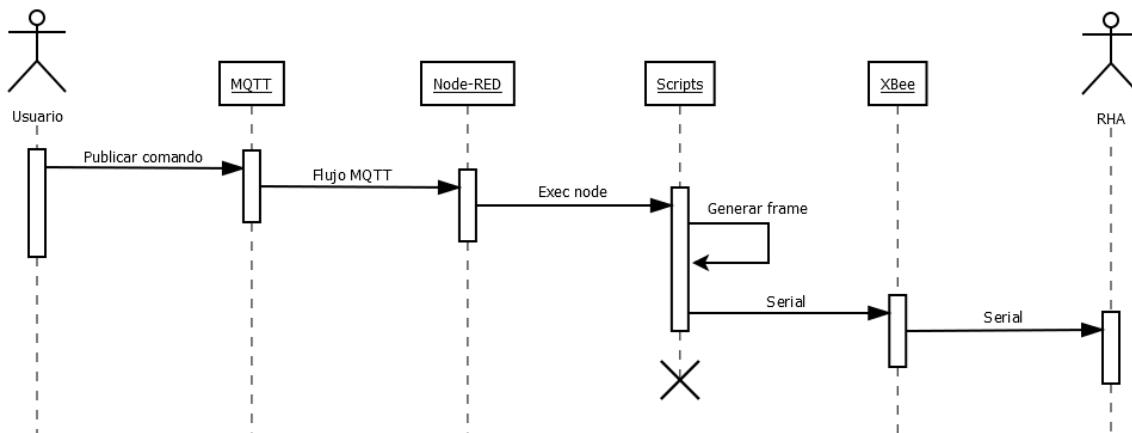


Figura 4.3: Diagrama de Secuencia de envío

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

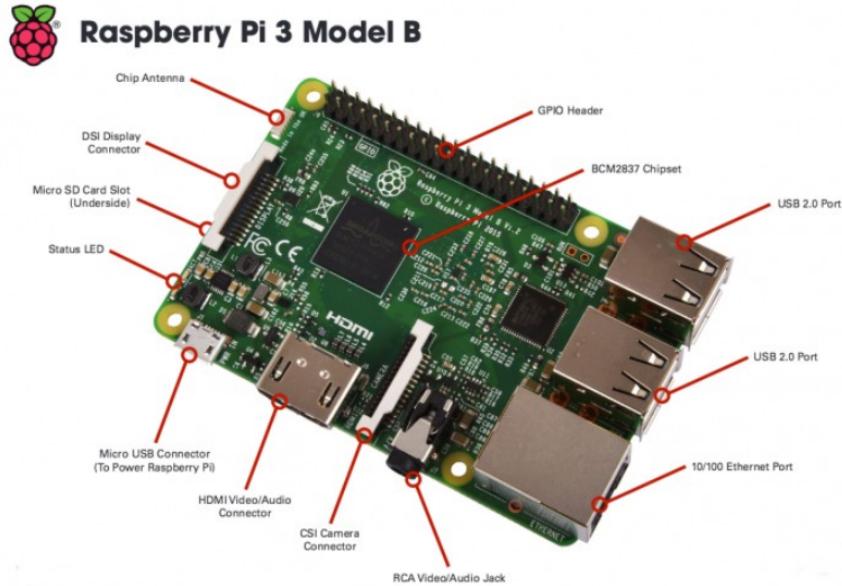


Figura 4.4: Layout de puertos de la Raspberry Pi

- 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

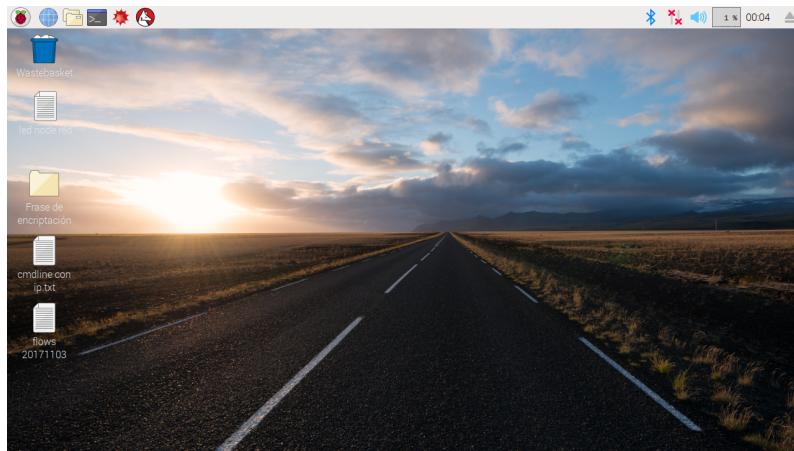


Figura 4.5: Escritorio de Raspbian

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*

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.

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

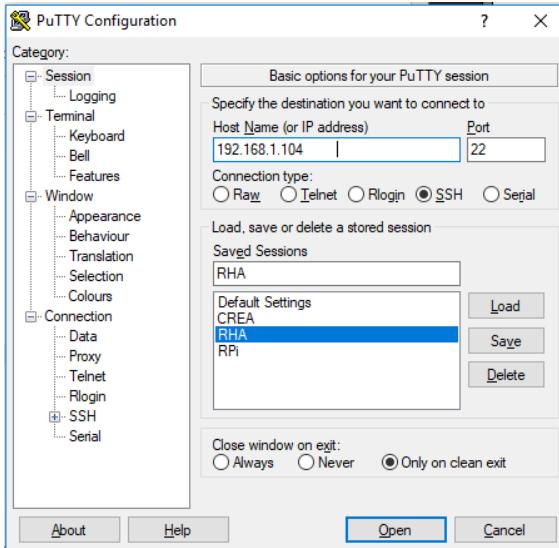


Figura 4.6: Interfaz de Putty

■ 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).

```
$ 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) [11] 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.

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

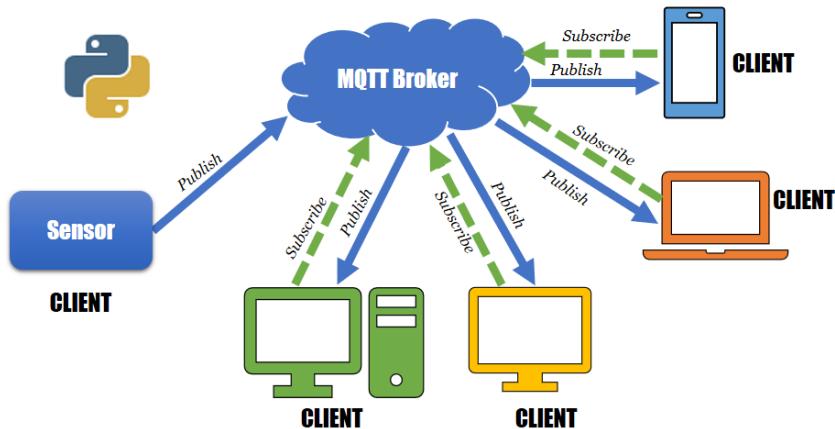


Figura 4.7: Arquitectura MQTT

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 por defecto y otros opcionales customizados. 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:

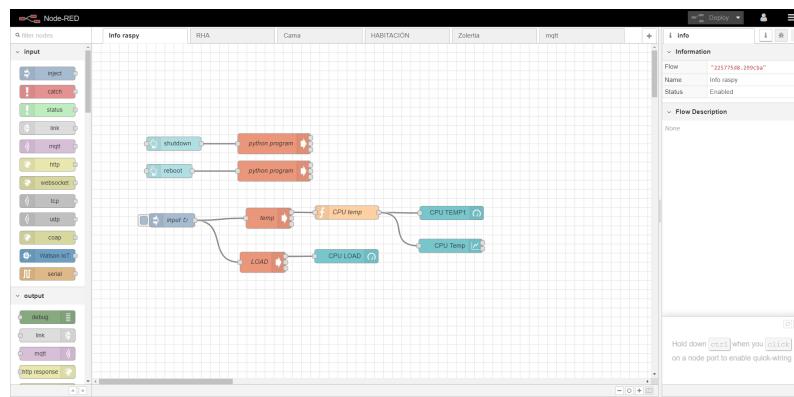


Figura 4.8: Ejemplo de interfaz de Node-RED

■ 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.

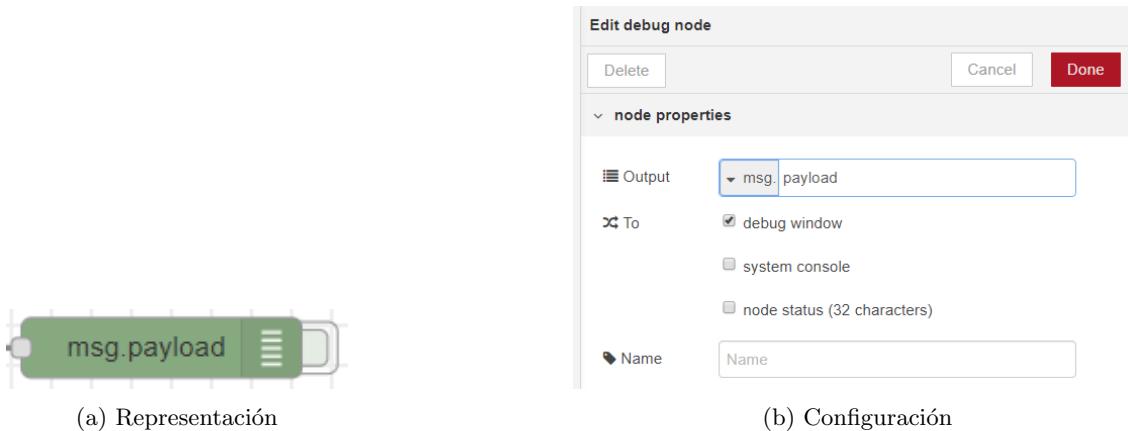


Figura 4.9: Nodo Debug

■ 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.

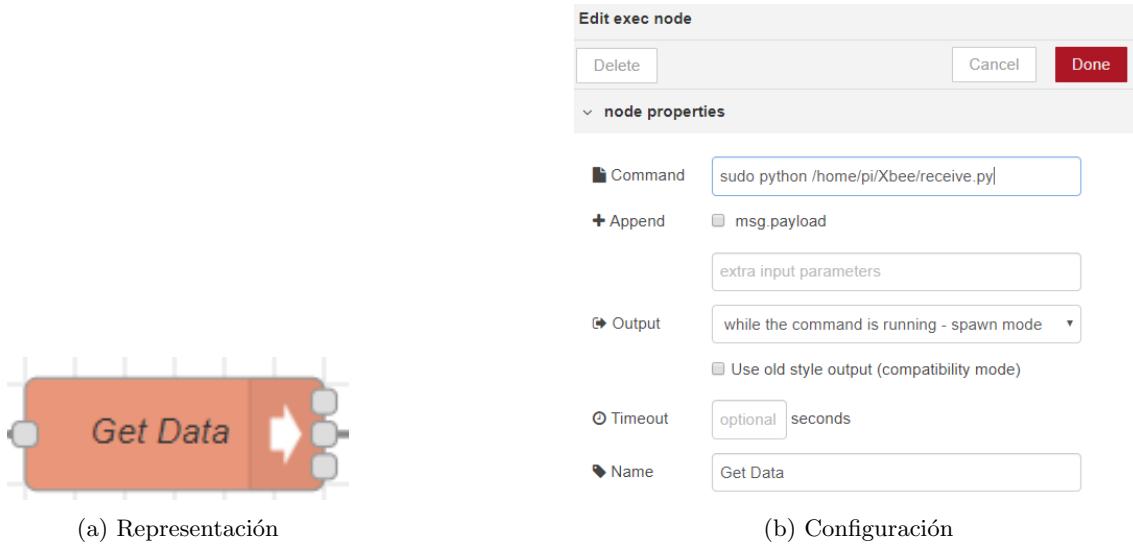


Figura 4.11: Nodo Debug

■ 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.

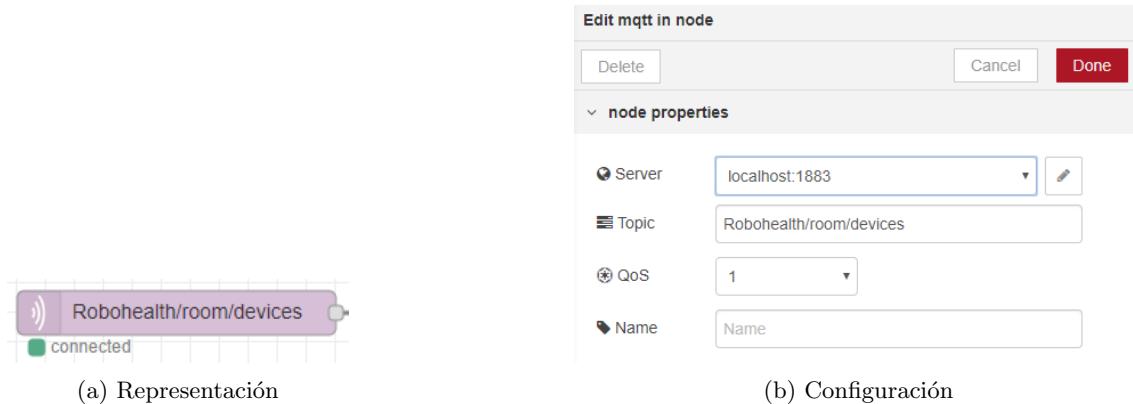


Figura 4.12: Nodo entrada MQTT

■ 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.

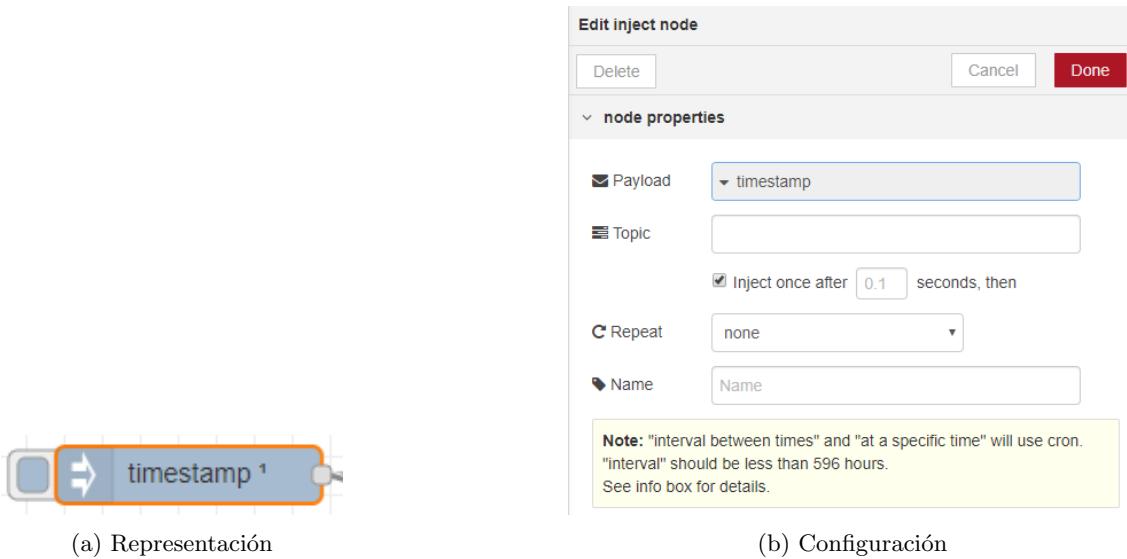


Figura 4.13: Nodo Inject

– Delay node

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

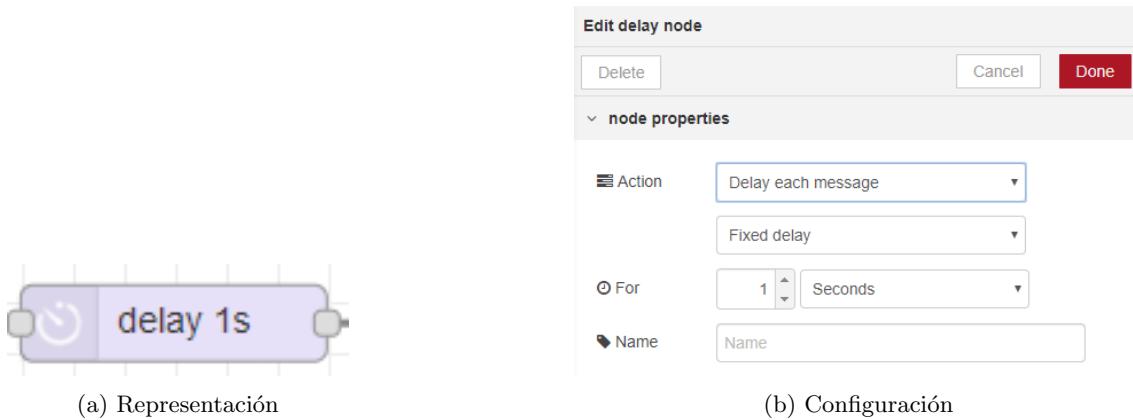


Figura 4.14: Nodo Delay

– 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*.

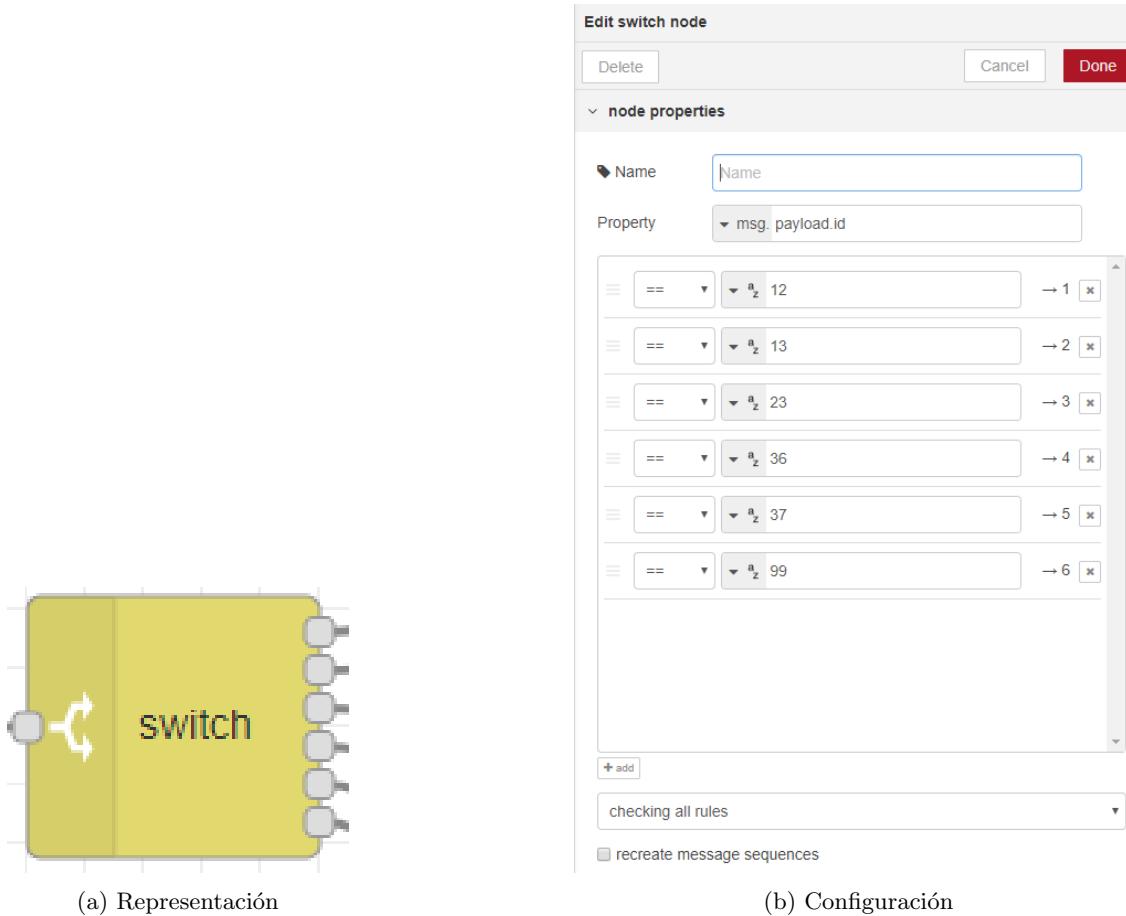


Figura 4.15: Nodo Switch

■ 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.

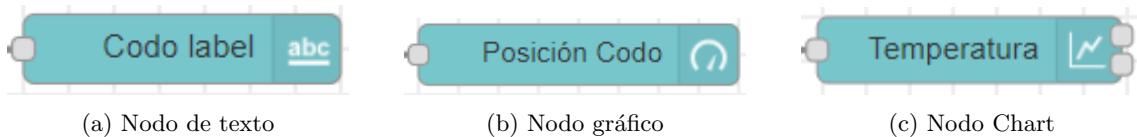


Figura 4.17: Nodos de salida

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.

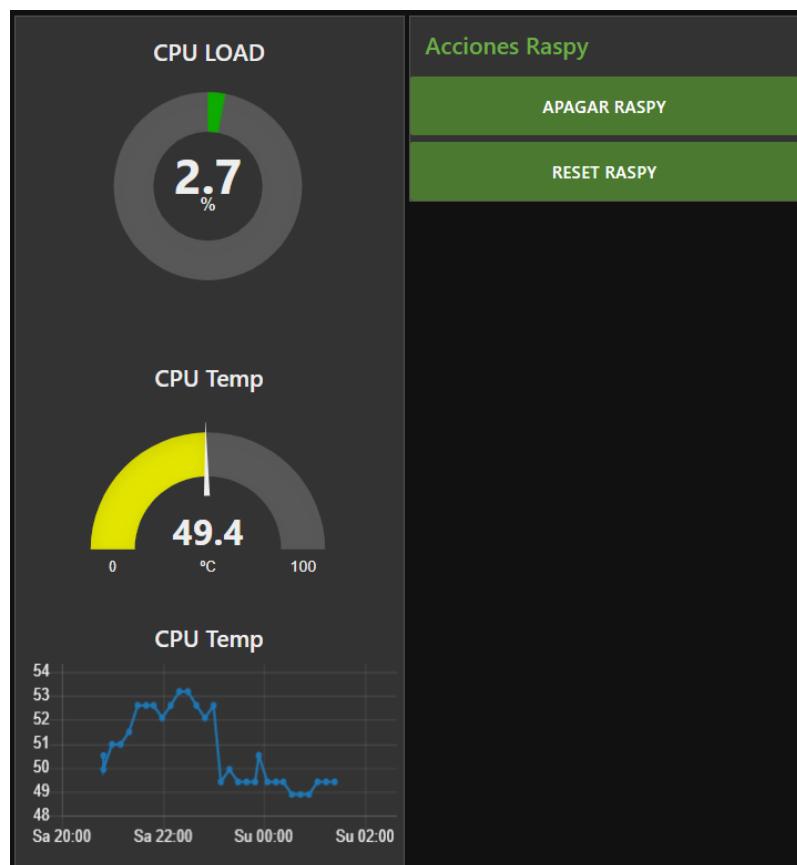


Figura 4.18: UI de control elemental

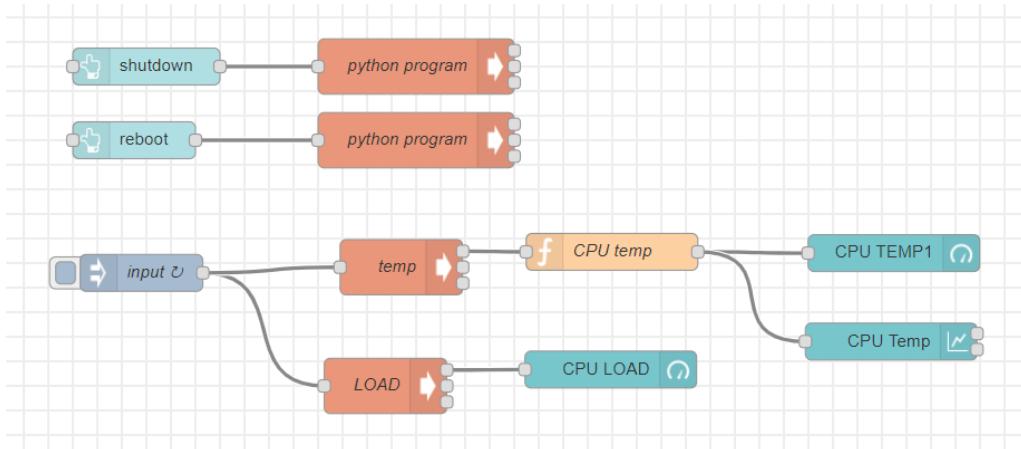


Figura 4.19: Flujo de control elemental



Figura 4.20: Dashboard

El flujo (figura 4.19) se basa en la ejecución de comandos para obtener la información de temperatura (línea 1/código 4.2.3) y carga de la CPU (línea 2/código 4.2.3). De igual manera, se lanzan comandos para apagar (línea 3/código 4.2.3) y reiniciar (línea 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
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).

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

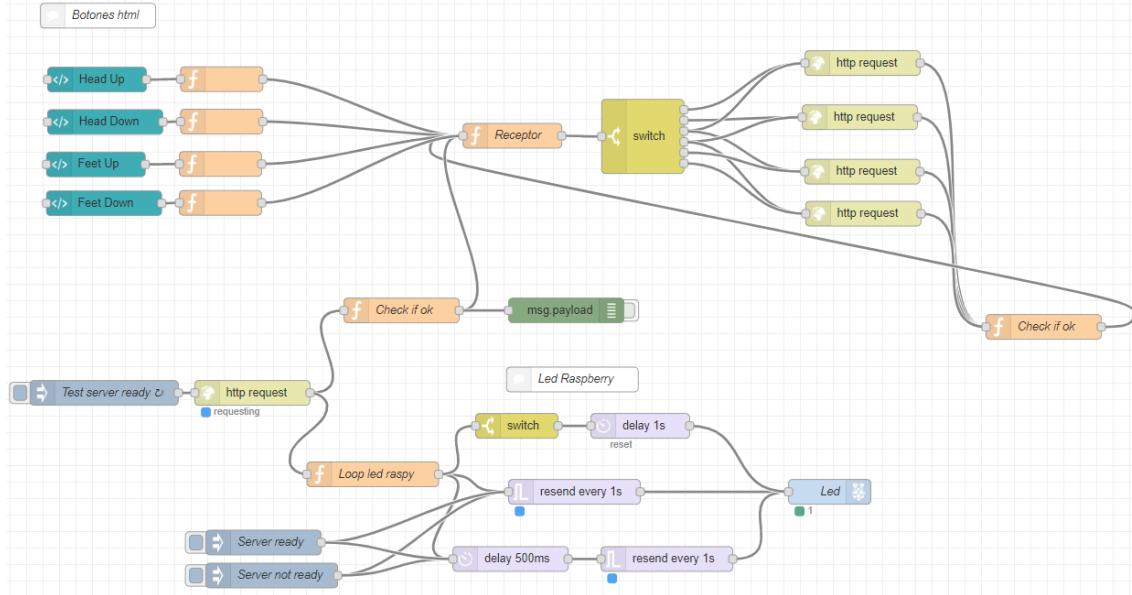


Figura 4.21: Flujo de control de la cama

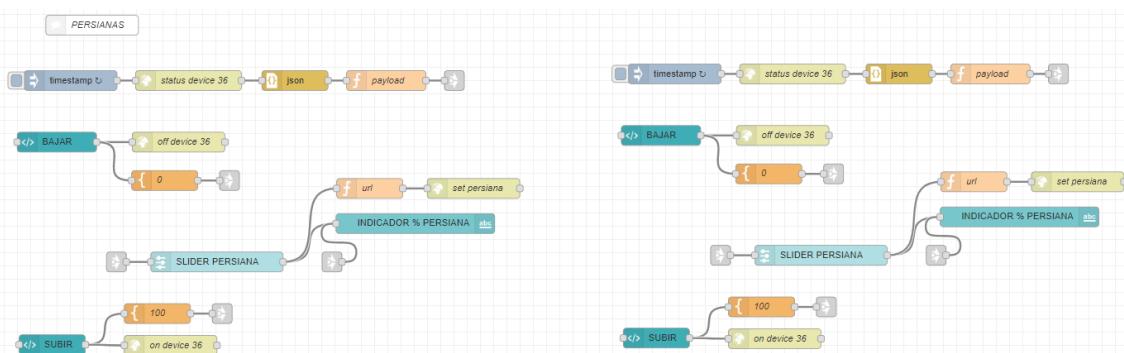


Figura 4.22: Flujo de control de las persianas

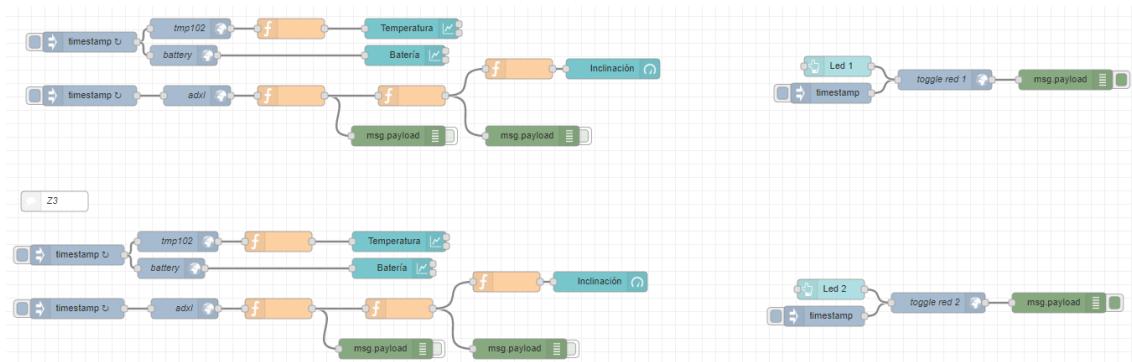


Figura 4.23: Flujo de control de Zolertia

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

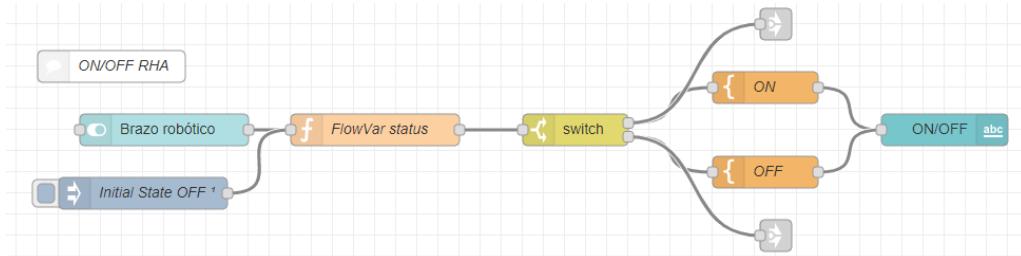


Figura 4.24: Flujo de encendido de RHA

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");
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

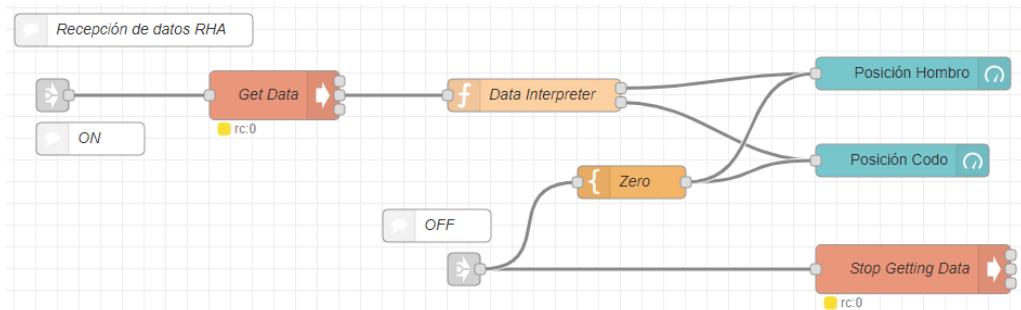


Figura 4.25: Flujo de recepción de RHA

La recepción de un flanco de encendido, pone a correr el comando 4.2.3 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

```
$ 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

```
$ 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.

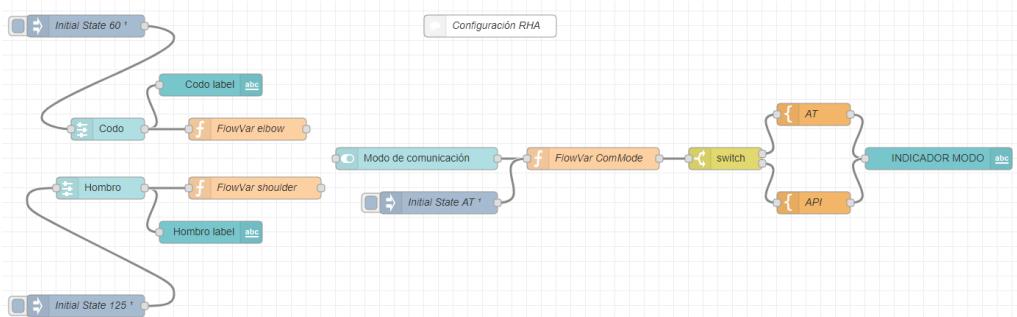


Figura 4.26: Flujo de configuración de RHA

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.

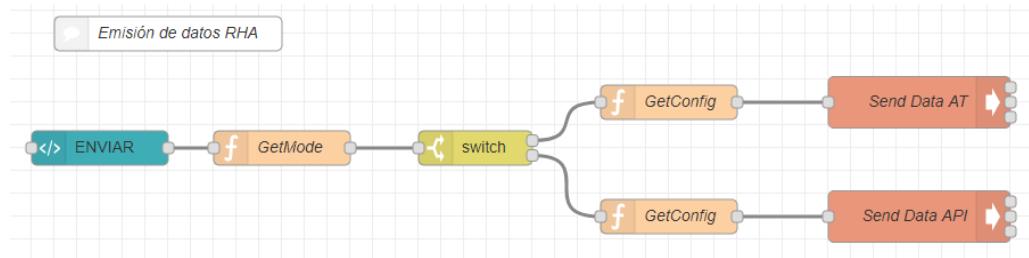


Figura 4.27: Flujo de emisión de RHA

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 hex_posX = posX . toString ( 16 );
4 | var hex_posY = posY . toString ( 16 );
5 | msg . payload = [ hex_posX , hex_posY ];
6 |

```

Los comandos para enviar los mensajes AT y API son los siguientes. En la práctica, *msg.payload* se añade al final del comando, indicando la información a enviar.

```

$ sudo python /home/pi/Xbee/sendAT.py
$ sudo python /home/pi/Xbee/sendAPI.py

```

El resultado gráfico de todos estos nodos y flujos se plasma en una interfaz de usuario como la de la figura 4.28.

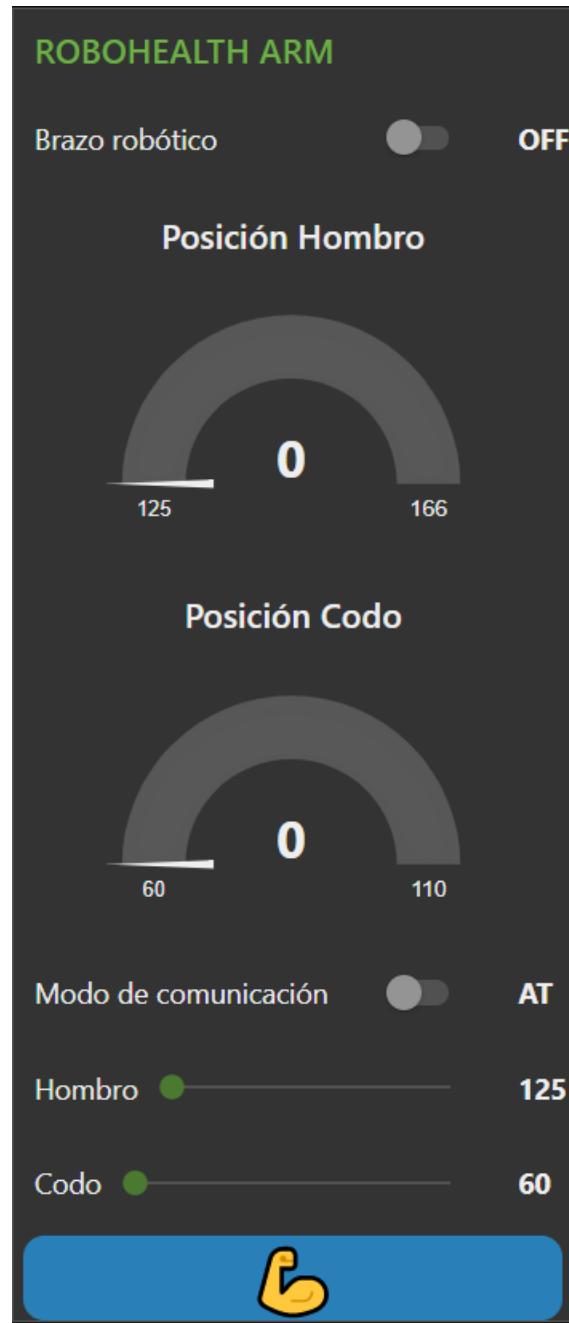


Figura 4.28: Dashboard para el control de RHA

Se puede observar un switch de ON/OFF para habilitar o deshabilitar la emisión y recepción de información. Las coordenadas articulares de hombro y codo son representadas mientras se actualizan en directo. Hay otro switch para seleccionar el modo de transmisión de datos y, por último, dos sliders para seleccionar una posición objetivo y un botón para lanzar la información al brazo robótico.

4.3. Transmisión de la información

Para la transmisión de la información, se eligió utilizar tecnología basada en la radiofrecuencia con el objetivo de diversificar la naturaleza de los dispositivos presentes en la habitación del proyecto RoboHealth.

Dentro de la radiofrecuencia, existen varios tipos de dispositivos y plataformas que pueden ayudarnos en nuestro objetivo. Un resumen de estas alternativas con sus características puede ser encontrado en la tabla 4.1.

Nombre	Características	
XBee	Pros	Mucha documentación, variedad de versiones, precio contenido, versatilidad
	Cons	Sobredimensionado
Jennic JN5148	Pros	Menor tamaño, coste contenidos
	Cons	Difícil distribución, documentación limitada
HPZB01W	Pros	Precio contenido
	Cons	Comunicación lenta, difícil manejo y escasa documentación
RFM12B	Pros	Coste muy reducido
	Cons	Escasa potencia, muy pequeña versatilidad

Tabla 4.1: Alternativas de radiofrecuencia

Finalmente, se decidió optar por los módulos XBee. Además de que sus características los hacen perfectamente funcionales para el desarrollo del presente proyecto y las alternativas no eran especialmente buenas, el departamento ya contaba con un par de estos módulos.

4.3.1. Módulos XBee

Los módulos XBee [18] (figura 1.5) son un conjunto de pequeños productos orientados a solucionar redes inalámbricas para la comunicación entre dispositivos. Entre sus características se encuentran:

- Gran variedad de productos que recorren un rango muy amplio tanto de consumos de potencia como de alcances.
- Son módulos que ofrecen un alto tráfico de datos
- Baja latencia en la comunicación
- Precisan una sincronización de comunicación predecible
- Su aplicación principal es la creación de redes *punto a punto* o *punto a multipunto*.

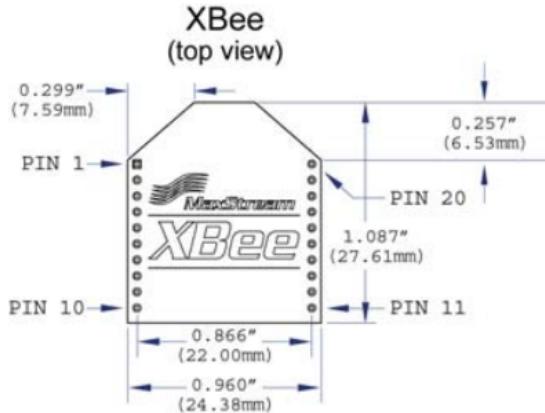


Figura 4.29: Layout del XBee S2

Los módulos XBee trabajan sobre la especificación **Zigbee** [20], que proporciona una alianza y un estandar sobre el que trabajar en redes *mesh* de comunicación. Esta especificación se basa en el estándar IEEE 802.15.4 [9], que define características físicas como el mecanismo DSSS para la modulación en radiofrecuencia (Se pueden encontrar más detalles en la sección 2.2).

Los módulos XBee de los que se disponen son los denominados XBee S2[19], permiten dos modos de comunicación (apartado 4.3.1) y una infinidad de parámetros configurables (apartado 5.2). Estos módulos sólo funcionan con módulos de la misma Serie 2, por lo que todos los dispositivos miembros de la red deben incluir este tipo de módulos.

El módulo XBee S2 trabaja a 3.3V de tensión de alimentación, el layout físico puede consultarse en la figura 4.29 y su pinout puede encontrarse en la figura 4.30.

Toda la información relacionada con los módulos de radiofrecuencia de Digi puede consultarse en el PDF destinado a ello[4] que puede encontrarse en internet.

Modos de comunicación

Como se ha comentado con anterioridad, existen dos modos de comunicación que pasan a desarrollarse a continuación. Es destacable el hecho de que los modos de comunicación deben coincidir entre los extremos comunicados para que la interpretación o el propio proceso de comunicación sea efectivo.

■ Modo AT

En el modo AT o modo transparente, el módulo actúa como una simple sustitución de la comunicación serial.

Cuando cualquier tipo de información es recibida por radiofrecuencia, es directamente trasladada al pin DOUT del módulo.

Cuando se recibe información a través del pin DIN, se transmite por radiofrecuencia directamente a las direcciones definidas en la configuración (apartado

Pin #	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART Data Out
3	DIN / <u>CONFIG</u>	Input	UART Data In
4	DIO8	Either	Digital I/O 8
5	<u>RESET</u>	Input	Module Reset (reset pulse must be at least 200 ns)
6	PWM0 / RSSI / DIO10	Output	PWM Output 0 / RX Signal Strength Indicator / Digital IO
7	PWM / DIO11	Either	Digital I/O 11
8	[reserved]	-	Do not connect
9	DTR / SLEEP_RQ / DI8	Input	Pin Sleep Control Line or Digital Input 8
10	GND	-	Ground
11	DIO4	Either	Digital I/O 4
12	<u>CTS</u> / DIO7	Either	Clear-to-Send Flow Control or Digital I/O 7
13	ON / <u>SLEEP</u>	Output	Module Status Indicator
14	[reserved]	-	Do not connect
15	Associate / DIO5	Either	Associated Indicator, Digital I/O 5
16	<u>RTS</u> / DIO6	Either	Request-to-Send Flow Control, Digital I/O 6
17	AD3 / DIO3	Either	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Either	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Either	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0	Either	Analog Input 0 or Digital I/O 0

Figura 4.30: Pinout del XBee S2

Encabezado	Longitud	Tipo	Datos									Checksum
1	2	3	4	5	6	7	8	9	...	n	n+1	
0x7E	MSB	LSB	Tipo	Información específica del tipo de Frame								

Tabla 4.2: Estructura de un API Frame

5.2).

Este método es el más simple pero, si en la red existen más de dos módulos XBee y se quiere discriminar la transmisión de información entre uno y otro, los módulos requieren ser reconfigurados a través de los registros de las direcciones de destino. En ese caso, limita mucho la facilidad de la operación.

Se puede comprobar que el script SendAT.py (apartado 5.1.1) envía exclusivamente los bytes correspondientes al protocolo de comunicación RHA. La recepción (apartado 5.1.3) también está pensado para recibir en modo AT, ya que se trata de ser lo menos intrusivo posible en el código fuente del brazo robótico y éste no está preparado para transmitir en modo API.

■ Modo API

El modo API (*Application Programming Interface*) es una alternativa al modo transparente que enmarca la emisión de datos en una serie de mensajes estandarizados. La tabla 4.2 recoge la estructura de un API Frame. Todos los mensajes usados a través de este modo deberán atenerse a esta estructura.

En la figura 4.31 se pueden ver los tipos de frames estandarizados para la transmisión de datos. De igual manera, los tipos de frames de recepción están recogidos en la figura 4.32.

API ID	Frame name	Description
0x88	AT Command Response	Displays the response to previous AT command frame
0x8A	Modem Status	Displays event notifications such as reset, association, disassociation, and so on.
0x8B	Transmit Status	Indicates wireless data transmission success or failure
0x90	Receive Packet	Sends wirelessly received data out the serial interface (AO = 0)
0x91	Explicit Rx Indicator	Sends wirelessly received data out the serial interface when explicit mode is enabled (AO 0)
0x92	IO Data Sample Rx Indicator	Sends wirelessly received IO data out the serial interface
0x94	XBee Sensor Read Indicator	Sends wirelessly received sensor sample (from a Digi 1-wire sensor adapter) out the serial interface
0x95	Node Identification Indicator	Displays received node identification message when explicit mode is disabled (AO = 0)
0x97	Remote AT Command Response	Displays the response to previous remote AT command requests
0x98	Extended Modem Status	Displays what is happening during the association when Verbose Join is enabled (DC10)
0xA0	Over-the-Air Firmware Update Status	Provides a status indication of a firmware update transmission attempt
0xA1	Router Record Indicator	Displays the multiple route hops after a Zigbee route record command
0xA3	Many-to-One Route Request Indicator	Indicates a many-to-one route request is received
0xA5	Join Notification Status	Indicates a module attempts to join, rejoin, or leave the network

Figura 4.32: API Frames de recepción

Para que un módulo configurado en este modo pueda emitir datos por radio, la información recibida por DIN debe adecuarse a los frames preestablecidos.

Si bien este modo aumenta la complejidad de los frames transmitidos, tiene la ventaja de contener la dirección de destino en el propio frame. Esto permite filtrar los receptores del mensaje en el propio mensaje, sin necesidad de acceder a la configuración del módulo XBee. Además, es posible recibir feedback de la transmisión del mensaje ya que la recepción de unos tipos de mensaje provocan la emisión de vuelta de otros que indican su recepción.

En el script sendAPI.py (apartado 5.1.2) se genera un API frame de tipo 0x10 *Transmit Request*.

Funciones de los módulos

Existen tres posibles funciones de cada módulo XBee dependiendo de la posición que ocupen en la red domótica. Cada función de XBee implica ligeros cambios en el firmware del módulo. En la figura 4.33 se puede observar un ejemplo de red domótica usando XBee.

■ Coordinador

El coordinador es el nodo central de la red. Su presencia en la red domótica es obligatoria y única (sólo se permite un nodo coordinador en la red). Se encarga de gestionar el resto de dispositivos, conecta y desconecta dispositivos de la red, asigna direcciones, traza mandatos y gestiona el consumo de energía pudiendo

API ID	Frame name	Description
0x08	AT Command	Queries or sets parameters on the local XBee
0x09	AT Command Queue Parameter Value	Queries or sets parameters on the local XBee without applying changes
0x10	Transmit Request	Transmits wireless data to the specified destination
0x11	Explicit Addressing Command Frame	Allows Zigbee application layer fields (endpoint and cluster ID) to be specified for a wireless data transmission
0x17	Remote AT Command Request	Queries or sets parameters on the specified remote XBee module
0x21	Create Source Route	Creates a source route in the module
0x24	Register Joining Device	Registers a module with the Trust Center

Figura 4.31: API Frames de transmisión

dormir al resto de dispositivos. Nunca puede apagarse o ponerse en modo de ahorro de energía.

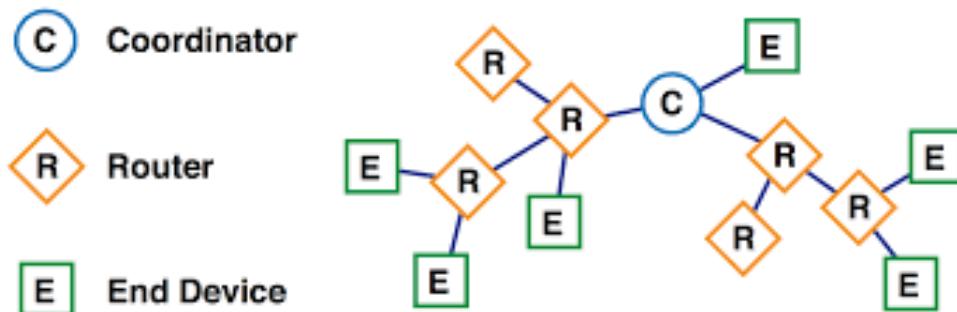


Figura 4.33: Ejemplo de red XBee

En el caso del presente proyecto, se tratará del nodo unido a la Raspberry Pi, que funciona como unidad de mando.

■ Router

El Router es un nodo con funciones completas, puede mandar y recibir información. Puede unirse a redes existentes y se usa, principalmente, para extender la red domótica más alla del alcance del Coordinador. Se encarga de ser mensajero trasladando información desde el coordinador hasta *End devices* u otros Routers; pudiendo llegar a modificar esta información.

Efectivamente, puede haber más de un router en la red y se encargan de gestionar los *End devices* que tengan conectados, pudiéndoles poner en modo ahorro de energía. No pueden ponerse en modo ahorro de energía y deben estar encendidos todo el tiempo, como el coordinador

En el caso de la conexión de RHA, el XBee del brazo robótico tomará la función de router ya que precisa de una comunicación bidireccional con el Coordinador.

■ End Device

Los *End Devices* son nodos extremos que pueden unirse a una red preexistente pero no pueden gestionar ningún dispositivo de ella. Pueden ponerse a ellos mismos en distintos modos de ahorro de energía. Las redes pueden tener un número muy alto de *End Devices*, tantos como direcciones sean almacenables en el nodo coordinador.

Su uso suele estar orientado al envío de la medida de algún sensor.

4.3.2. XBee Shield

La XBee Shield (figura 1.4) es una solución que permite una interacción sencilla con los módulos XBee. Para ello, se basa en el uso de Arduino UNO (figura 1.3. La XBee Shield posee un zócalo para situar el módulo XBee, está diseñada para encajar a la perfección sobre el Arduino UNO y esto descubre multitud de opciones.

Los diseños, layouts y planos son libres y pueden ser encontrados en internet. En el Anexo C.1 se localiza el esquemático de la XBee Shield.

Posee dos jumpers selectores que establecen el modo en el que debe comportarse la XBee Shield. El hecho de que haya dos es simplemente para asegurar un modo, ambos jumpers deben situarse en la misma posición para un correcto funcionamiento. La selección de estos modos configura la conexión entre la comunicación serial del módulo XBee y la comunicación serial entre el microcontrolador y el integrado USB-to-Serial de la Arduino UNO.

Modo XBee

Una de las posiciones de los jumpers es la posición XBee. En esta posición, el pin DOUT del módulo XBee se conecta al RX del microcontrolador y, por lo tanto, al TX del USB. Por otro lado, el pin DIN estaría conectado al TX del microcontrolador y, consecuentemente, al RX del USB.

Esta configuración permite que aquello que envíe el microcontrolador sea transmitido tanto por USB al ordenador como por radiofrecuencia. Sin embargo, la recepción de datos por parte del microcontrolador se realizará únicamente desde el módulo XBee, quedando inhabilitada el envío de datos por serial desde un ordenador.

Lógicamente, ya que se desea una comunicación bidireccional de datos con una Raspberry Pi, esta configuración queda descartada.

Modo USB

El otro modo alcanzable a través de los jumpers es el modo USB. Este modo consiste en la comunicación directa del pin DOUT del XBee al RX del USB y, por su parte, la conexión del pin DIN al TX del USB.

Esto posibilita la conexión directa del módulo XBee al ordenador siempre que ningún otro dispositivo interfiera en la comunicación. Este dispositivo puede ser, perfectamente, el microcontrolador. Es esta la razón por la que es necesario retirar el microcontrolador del Arduino UNO para que funcione de manera adecuada el modo USB⁵. Para retirar el microcontrolador, es necesario que la versión de Arduino UNO utilizada use el microcontrolador en su empaquetado de orificio pasante colocado sobre un zócalo. La versión SMD requiere desoldarlo y no termina de ser cómodo.

Puesto que en ninguno de los puntos del proyecto requerimos del preprocessamiento de los datos por un Arduino externo, el microcontrolador no aporta ninguna utilidad. Es la configuración USB, por lo tanto, la usada a lo largo del proyecto.

En concreto, hay una situación en la que este modo adquiere especial sentido: a la hora de configurar el módulo y, en general, al usar XCTU con un módulo XBee.

⁵De lo contrario, el Arduino UNO funcionaría de manera normal, ignorando el Modulo XBee

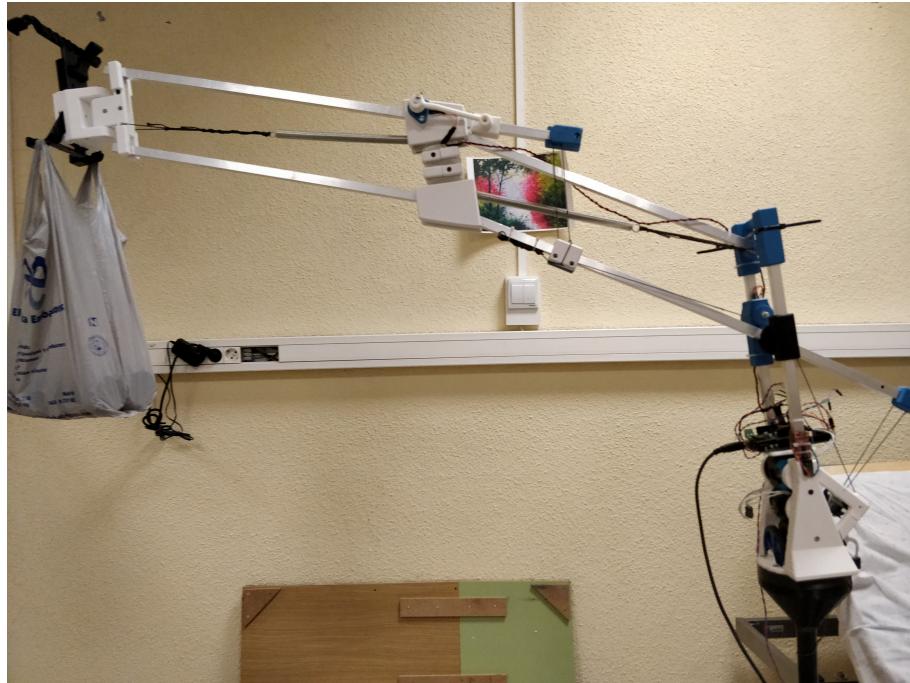


Figura 4.34: RoboHealth Arm

Se precisa una conexión directa Xbee-ordenador que la Xbee Shield en modo USB puede proporcionar.

4.4. RoboHealth Arm

RoboHealth Arm[5] (figura 4.34) es un brazo robótico de tres grados de libertad cuyo sistema de acción se basa en un mecanismo controlado por una serie de cuerdas que son recogidas o liberadas por unos servos. Unos potenciómetros realimentan la posición de las dos articulaciones que funcionan con este mecanismo. El tercer grado de libertad, encargado de la rotación sobre su eje, no funciona de igual manera, sino que lo hace a través de un engranaje simple sin haber forma de realimentar su posición. Por lo tanto, el control de este tercer eje es absoluto.

El programa que controla el brazo robótico[7] se carga sobre un microcontrolador Arduino Mega

RoboHealth Arm se trata de un Trabajo Final de Grado[5] realizado previamente en el marco del proyecto RoboHealth.

4.4.1. Protocolo de comunicación RHA

RoboHealth Arm incluye en su desarrollo un protocolo de comunicación detallado en [5]/Anexo E. Los mensajes enmarcados en este protocolo siguen la estructura recogida en la tabla 4.3

Encabezado		Longitud	Tipo	Parámetros					Checksum
1	2	3	4	5	6	7	...	n	n+1
0xFF	0xFF	Longitud	Tipo	Parámetros del mensaje					Byte de Checksum

Tabla 4.3: Estructura de un paquete genérico de comunicación RHA

En general, este proyecto se centra en dos tipos de mensajes.

■ Frame de estado

Mensaje enviado de forma periódica desde el brazo informando de su estado. Como se puede apreciar en la estructura del frame recogida en la tabla 4.5, permite conocer posición, velocidad y par de cada articulación.

Byte	Información contenida
0	Encabezado (Fijo: 0xFF)
1	Encabezado (Fijo: 0xFF)
2	Longitud (0x06)
3	Tipo (0x02)
4	Objetivo de posición para la primera articulación
5	Objetivo de posición para la segunda articulación
6	Objetivo de posición para la tercera articulación
7	<i>Checksum</i>

Tabla 4.4: Estructura del frame de comando[5]

■ Frame de comando

Es el mensaje utilizado para comandar el brazo robótico. RHA está permanentemente leyendo la entrada serial. La codificación de este tipo de mensaje se muestra en la tabla C.1.

Byte	Información contenida
0	Encabezado (Fijo: 0xFF)
1	Encabezado (Fijo: 0xFF)
2	Longitud a leer (18)
3	Tipo de mensaje (0)
Información de la primera articulación	
4	Posición articular
5	Velocidad (con dirección). Byte L
6	Velocidad (con dirección). Byte H
7	Par aplicado (con dirección). Byte L
8	Par aplicado (con dirección). Byte H
Información de la segunda articulación	
9	Posición articular
10	Velocidad (con dirección). Byte L
11	Velocidad (con dirección). Byte H
12	Par aplicado (con dirección). Byte L
13	Par aplicado (con dirección). Byte H
Información de la tercera articulación	
14	Posición articular
15	Velocidad (con dirección). Byte L
16	Velocidad (con dirección). Byte H
17	Par aplicado (con dirección). Byte L
18	Par aplicado (con dirección). Byte H
19	<i>Checksum</i>

Tabla 4.5: Estructura del frame de estado[5]

Capítulo 5

Implementación del proyecto

5.1. Scripts en la Raspberry Pi

En la Raspberry Pi están guardados una serie de 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.

A continuación se explica detalladamente cada uno de estos scripts.

5.1.1. SendAT.py

Es un script encargado de enviar comandos en modo AT hacia el puerto serial adecuado. El **código íntegro de SendAT.py** se puede encontrar en el Anexo B.1.

En las primeras 5 líneas se importan las librerías necesarias. Se puede observar que hacemos uso de la librería del sistema, para acceder a los atributos pasados con el comando; a la librería de tiempo, relacionada con los timestamps del logging de la información realizado a través de la librería *logging*; y a la librería *serial* para poder acceder a esos puertos.

En la línea 7 se configura el log de la información. Se establece el archivo */home/pi/Xbee/SendLogs/sentAT.log* como archivo de destino, se desactivan restricciones de nivel¹ y, por último, se determina el formato del mensaje que incluye un timestamp al inicio.

Entre las líneas 9 y 16 se abre una comunicación serial con el puerto */dev/tt-*

¹Las restricciones de nivel son barreras a la introducción de ciertos tipos de mensajes en el archivo de destino en función de su importancia (ERROR/WARNING/INFO)

y *ACM0*² a una tasa de baudios de 115200. El resto de la configuración entra dentro de los parámetros usuales y por defecto.

De la línea 18 a 22 se comprueba que el número de argumentos pasados es el adecuado. De lo contrario, se simulan unos argumentos neutrales genéricos y se guarda un *Warning* en el archivo de log. Si los argumentos se han pasado de manera correcta, se sacan del atributo del sistema.

En las líneas 24-27 se obtienen los números enteros a partir de los argumentos hexadecimales de entrada.

En la línea 28 se calcula el checksum de acuerdo al protocolo de comunicación de RHA.

A partir de la línea 30, se comprueba que los valores introducidos están dentro del rango admisible para el brazo robótico, se integra un frame y se escribe por serial; guardando el resultado de la operación en el archivo de log.

Las características del modo AT se explican en el apartado 4.3.1.

5.1.2. SendAPI.py

Se trata de un script que, de igual manera que SendAT.py, envía datos hacia un puerto serial, pero esta vez encuadrado dentro del modo API. Este modo modifica la estructura del frame enviado. El **código íntegro de SendAPI.py** puede ser encontrado en el Anexo B.2.

El código es similar al de SendAT.py. De hecho, hasta la línea 29, es exactamente igual a excepción de configurar el archivo */home/pi/Xbee/SendLogs/sentAPI.log* como archivo de destino del log de información.

En la línea 30, se encuentra el cálculo del checksum de necesario para el modo API.

Entre las líneas 32 y 41 se forma el frame de acuerdo a las necesidades API y se envía por serial; guardando el resultado en el log.

Las características del modo API y sus diferencias con el modo AT se explican en el apartado 4.3.1.

5.1.3. Receive.py

Receive.py es el script encargado de obtener los frames recibidos por serial. El **código íntegro de receive.py** puede ser encontrado en el Anexo B.3.

²Linux detecta como un dispositivo en un puerto *ttyACM** a aquellos dispositivos de comunicación USB del subtipo *Abstract Control Mode*. Arduino se encuentra en esta categoría

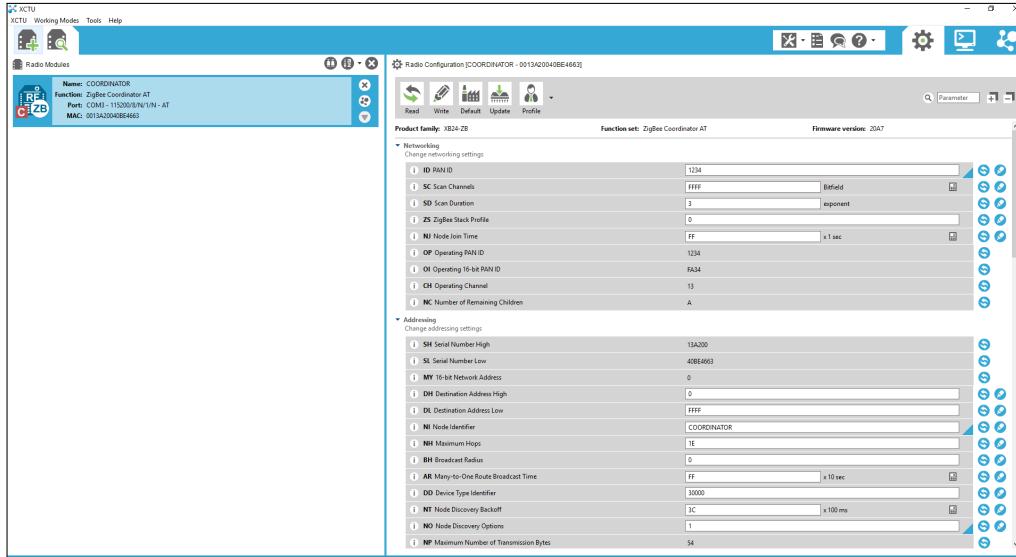


Figura 5.1: Interfaz de XCTU

Se configura el log de información con salida hacia el archivo `/home/pi/Xbee-/ReceiveLogs/receiveAT.log`. La configuración del puerto serial es igual a la indicada en los dos scripts anteriores.

Entre las líneas 19 y 25, se puede encontrar la función `serialReader()` que se encarga de leer un byte por serial y transformarlo a entero. Para este proceso, es necesario importar la librería `struct`, cuyo método `unpack()` sirve de mucha utilidad. Devuelve -1 si no se lee nada por serial.

A partir de la línea 27, se puede observar un bucle infinito que comprueba las lecturas por serial hasta dar con un frame del brazo. Una vez encontrado este frame, lo guarda en el archivo de log y extrae las posiciones articulares. Imprime estas dos posiciones como vector por la salida de error `stderr` por necesidades externas explicadas en el apartado 5.5.

5.2. Configuración de módulos XBee

Para configurar los módulos XBee, Digi proporciona un software llamado XCTU³.

Una vez conectado el dispositivo XBee, el primer paso es descubrirlo. Haciendo click en el ícono destinado a ello, seleccionando el puerto e indicando las opciones sobre las que escanear dispositivos se iniciará la búsqueda. Sólo queda añadir el dispositivo cuando aparezca en pantalla.

Una vez añadido, se cargará la configuración actual en la interfaz de Node-RED (figura 5.1) y se podrán visualizar y modificar todos los parámetros.

Para cambiar el firmware, se puede hacer click en *Update* (figura 5.2) y seleccionar

³Puede descargarse desde <https://www.digi.com/products/iot-platform/xctu>

un nuevo firmware y versión. Es en esta pantalla en la que se puede cargar firmware orientado a un modo de comunicación en concreto (AT o API). Al hacer click en Update comenzará el proceso de escritura del firmware y, tras unos segundos, el módulo estará listo para usarse.

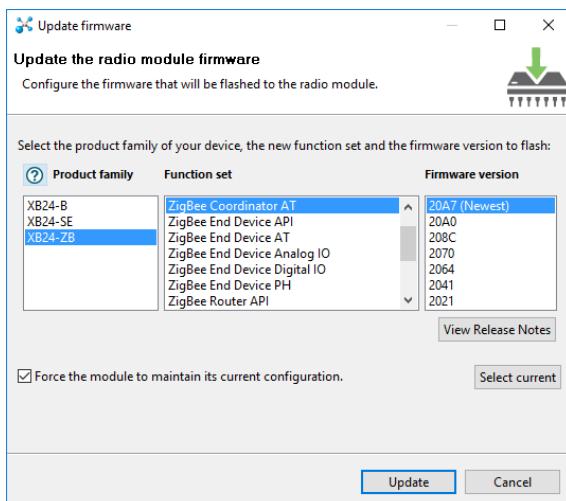


Figura 5.2: Carga de firmware en XCTU

La interfaz de XCTU permite modificar multitud de parámetros que deben coincidir en todos los extremos de la comunicación. En la figura 5.3 se pueden observar los parámetros de configuración del XBee Coordinador AT, mientras que en la figura 5.4 está la configuración del Router AT. En modo API son similares

5.2. CONFIGURACIÓN DE MÓDULOS XBEE

59

Product family: XB24-ZB **Function set:** ZigBee Coordinator AT **Firmware version:** 20A7

- ▼ Networking

Change networking settings

i ID PAN ID	1234
i SC Scan Channels	FFFF Bitfield
i SD Scan Duration	3 exponent
i ZS ZigBee Stack Profile	0
i NJ Node Join Time	FF x 1 sec
i OP Operating PAN ID	1234
i OI Operating 16-bit PAN ID	FA34
i CH Operating Channel	13
i NC Number of Remaining Children	A
- ▼ Addressing

Change addressing settings

i SH Serial Number High	13A200
i SL Serial Number Low	40BE4663
i MY 16-bit Network Address	0
i DH Destination Address High	0
i DL Destination Address Low	FFFF
i NI Node Identifier	COORDINATOR
i NH Maximum Hops	1E
i BH Broadcast Radius	0
i AR Many-to-One Route Broadcast Time	FF x 10 sec
i DD Device Type Identifier	30000
i NT Node Discovery Backoff	3C x 100 ms
i NO Node Discovery Options	1
i NP Maximum Number of Transmission Bytes	54
i CR PAN Conflict Threshold	3
- ▼ ZigBee

Change ZigBee protocol addressing settings

i SE ZigBee Source Endpoint	E8
i DE ZigBee Destination Endpoint	E8
i CI ZigBee Cluster ID	11
- ▼ RF Interfacing

Change RF interface options

i PL Power Level	Highest [4]
i PM Power Mode	Boost Mode Enabled [1]
i PP Power at PL4	3
- ▼ Security

Change security parameters

i EE Encryption Enable	Disabled [0]
i EO Encryption Options	0 Bitfield
i KY Encryption Key	
i NK Network Encryption Key	
- ▼ Serial Interfacing

Change modem interfacing options

i BD Baud Rate	115200 [7]
i NB Parity	No Parity [0]
i SB Stop Bits	One stop bit [0]
i RO Packetization Timeout	3 x character times
i D7 DIO7 Configuration	CTS flow control [1]
i D6 DIO6 Configuration	Disable [0]
- ▼ AT Command Options

Change AT command mode behavior

i CT AT Command Mode Timeout	64 x 100ms
i GT Guard Times	3E8 x 1ms
i CC Command Sequence Character	2B Recommended: ...-0x7F (ASCII)
- ▼ Sleep Modes

Configure low power options to support end device children

i SP Cyclic Sleep Period	20 x 10 ms
i SN Number of Cyclic Sleep Periods	1
- ▼ I/O Settings

Figura 5.3: Propiedades del XBee Coordinador AT

Product family: XB24-ZB Function set: ZigBee Router AT Firmware version: 22A7

Networking
Change networking settings

i ID PAN ID	1234			
i SC Scan Channels	FFFF	Bitfield		
i SD Scan Duration	3	exponent		
i ZS ZigBee Stack Profile	0			
i NJ Node Join Time	FF	x 1 sec		
i NW Network Watchdog Timeout	0	x 1 minute		
i JV Channel Verification	Enabled [1]			
i JN Join Notification	Enabled [1]			
i OP Operating PAN ID	0			
i OI Operating 16-bit PAN ID	FFFF			
i CH Operating Channel	0			
i NC Number of Remaining Children	C			

Addressing
Change addressing settings

i SH Serial Number High	13A200			
i SL Serial Number Low	40BE4589			
i MY 16-bit Network Address	FFFE			
i DH Destination Address High	0			
i DL Destination Address Low	0			
i NI Node Identifier	ROUTER			
i NH Maximum Hops	1E			
i BH Broadcast Radius	0			
i AR Many-to-One Route Broadcast Time	FF	x 10 sec		
i DD Device Type Identifier	30000			
i NT Node Discovery Backoff	3C	x 100 ms		
i NO Node Discovery Options	0			
i NP Maximum Number of Transmission Bytes	54			
i CR PAN Conflict Threshold	3			

ZigBee Addressing
Change ZigBee protocol addressing settings

i SE ZigBee Source Endpoint	E8		
i DE ZigBee Destination Endpoint	E8		
i CI ZigBee Cluster ID	11		

RF Interfacing
Change RF interface options

i PL Power Level	Highest [4]		
i PM Power Mode	Boost Mode Enabled [1]		
i PP Power at PL4	3		

Security
Change security parameters

i EE Encryption Enable	Disabled [0]			
i EO Encryption Options	0	Bitfield		
i KY Encryption Key				

Serial Interfacing
Change modem interfacing options

i BD Baud Rate	115200 [7]			
i NB Parity	No Parity [0]			
i SB Stop Bits	One stop bit [0]			
i RO Packetization Timeout	3	x character times		
i D7 DIO7 Configuration	CTS flow control [1]			
i D6 DIO6 Configuration	Disable [0]			

AT Command Options
Change AT command mode behavior

i CT AT Command Mode Timeout	64	x 100ms		
i GT Guard Times	3E8	x 1ms		
i CC Command Sequence Character	2B	Recommended: ...-0x7F (ASCII)		

Sleep Modes
Configure low power options to support end device children

i SM Sleep Mode	No Sleep (Router) [0]		
-----------------	-----------------------	--	--

Figura 5.4: Propiedades del XBee Router AT

Entre los parámetros configurables más destacables se encuentran los siguientes:

- **PAN ID** es el identificador de la red domótica.
- **DH** y **DL** ponen límites a las direcciones entre las cuales el dispositivo transmitirá. Estas configuraciones a 0 en el router ponen al coordinador de la red como único destinatario.
- **NI** es el nombre identificador del nodo.
- **EE** maneja la posibilidad de encriptar la información para añadir una capa extra de seguridad.
- **BD, NB o SB** son los parámetros correspondientes a la comunicación serial. En este caso, se sitúa la tasa de baudios en 115200, que es el máximo admitible por XCTU.

Por último, queda comentar la posibilidad de usar XCTU como terminal serie de manera sencilla y adaptada a la aplicación que estamos comentando. En la pestaña de consola se puede consultar la emisión y recepción de cada módulo, como se muestra en la figura 5.5.

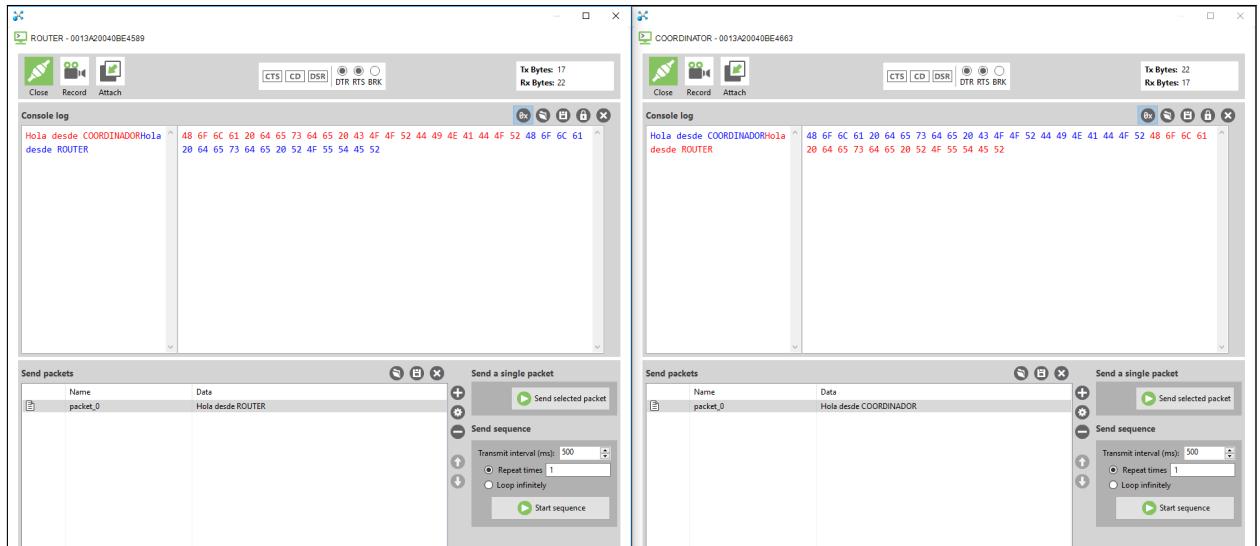


Figura 5.5: Interfaz de consola de XCTU

5.2.1. Perfiles de comunicación

La configuración introducida en XCTU se puede guardar y cargar de manera sencilla a través de la figura del perfil de comunicación. Estos perfiles son archivos que almacenan el firmware utilizado junto con toda la configuración específica.

Esto posibilita, como es el caso, la generación de los perfiles:

CoordinatorAPI.xpro

RouterAPI.xpro

CoordinatorAT.xpro

RouterAT.xpro

Siendo un proceso sencillo el de carga de un perfil a un módulo, es la manera más simple y rápida de cambiar el modo de comunicación de la red. No hay posibilidad de hacerlo sin cambiar el firmware.

5.3. Modificaciones a RHA

Se han realizado una serie de modificaciones al brazo robótico RHA con el fin de, por un lado, hacerlo compatible con las tecnologías usadas en este proyecto y, por el otro, solucionar algunos defectos que tenía.

En todo caso, la intención a lo largo del desarrollo del proyecto es obtener buenos resultados con una modificación mínima del estado inicial del brazo robótico, reduciendo al mínimo la posibilidad de generar incompatibilidades con futuros desarrollos ya planificados.

5.3.1. Modificaciones Hardware

Con modificaciones hardware se cuentan los cambios al proyecto inicial en cuanto a la configuración física, sensores o actuadores.

- El brazo robótico, de inicio, no tenía montado el mecanismo destinado al eje de rotación. Su funcionamiento era mejorable y este propósito se sale fuera del marco del presente proyecto, así que se decidió trabajar ignorando este eje.
- El servo encargado de mover la articulación correspondiente al codo del RHA reportaba un error de sobrecarga aunque se le intentara hacer funcionar en vacío. La sensación al forzar la rotación sin alimentar el servo era muy diferente a la del otro servo que funcionaba correctamente, era mucho más difícil provocar esa rotación con la mano. Por todo esto, se concluyó que el servo estaba defec-tuoso y se sustituyó físicamente por el servo hasta ese momento encargado del movimiento del eje de rotación que se había definido como inutilizado.
- Los potenciómetros no captaban de manera adecuada los movimientos de sus respectivos ejes. En el caso del codo, la pieza de impresión 3D que encajaba en el potenciómetro para rotarlo se había partido. En la articulación del hombro, la pieza de engranaje encargada de rotar el potenciómetro había cogido cierta holgura, provocando que no se generara rotación en la totalidad del movimiento del eje. En ambos casos se han solucionado los problemas mediante la utilización de un pegamento suficientemente fuerte.

5.3.2. Modificaciones Software

Teniendo en mente que la intención es reducir las modificaciones software lo máximo posible, solo se han hecho dos cambios de manera obligatoria.

- El defecto hardware que ha obligado a cambiar la posición de los servos tiene repercusiones en el software. Los servos no permiten otro orden de conexión más allá del previamente establecido. Más información sobre la comunicación del brazo con los servos puede encontrarse en el [5]/Anexo D. Si se cambia la posición física de los servos pero no se tocan sus conexiones, no hay más remedio que intercambiar la identificación de los servos en el software.
- Como se ha mencionado en el apartado 5.2, la tasa de baudios máxima configurable para la comunicación serial de los módulos XBee es 115200. En el brazo robótico, la interfaz encargada de mandar frames de estado y recibir frames de comando trabajaba a 250000. Este dato también ha tenido que ser modificado para habilitar la comunicación, haciéndolo coincidir con el baudaje de los módulos XBee.

Por otro lado, y siendo un añadido al software, es necesaria la modificación de las funciones sendPackage() y receivePackage() para usar el modo API de comunicación. Al enviar un frame de información, este debe ser enmarcado en un API Frame de tipo 0x10 *Transmit request* y eso es trabajo del software del brazo. En el mismo sentido va la modificación de la función encargada de la recepción de paquetes, debe esperarse la recepción de un API frame de manera previa a la recepción del frame propio de la comunicación con el RHA.

El código 5.1 es un ejemplo de esto. Preprocesa el API frame antes de meterse a interpretar el frame de comandos RHA. Debe entenderse que es un else if tras un condicional que lee el inicio de un frame RHA. Es, por tanto, una alternativa no limitante de la lectura en modo transparente.

Código 5.1: RHA API - getPackage()

```

1  else if (by1 == 0x7E)           //XBee communication
2  {
3      // Now a XBee msg can be read. XBee header was ok
4      //Serial.println("XBEE");
5      Serial_PYNTERFACE.read(); xbee_length = Serial_PYNTERFACE.read();
6      uint8_t *data = new uint8_t [length];
7      for(int k=0; k<length; k++) data[k]=Serial.read();
8      if(data[0] == 0x90) {
9          DebugSerialRRHALn("getPackage:_Receive_Packet");
10     }
11    else {
12        DebugSerialRRHALn("getPackage:_Not_supported_Xbee_frame");
13        return;
14    }
15
16    if (data[12] == 0xFF && data[13] == 0xFF) {
17        // Now a msg can be read. Header was ok
18        DebugSerialRRHALn("getPackage:_Header_ok");
19        Serial.println("getPackage:_Header_ok");
20        length = data[14]; checksum += length;
21        for (int k=0; k<length; k++) buffer_[k]=data[k+15];
22        // Check if checksum is correct

```

```

23         for (i = 0; i < length-1; i++) {
24             checksum += buffer_[i];
25         }
26         checksum = ~checksum;
27         // If its correct it can handle the information
28         // Checksum is the last byte in package
29         //if (readCount == length-1) { //&& checksum == buffer_[length]) {
30         if (buffer_[0] == PynterfaceConstants::ARTICULAR.GOAL) {
31             pynterface_goal_.x = buffer_[1];
32             pynterface_goal_.y = buffer_[2];
33             pynterface_goal_.z = buffer_[3];
34             goToArticularPos(pynterface_goal_);
35         }
36     }
37 }
```

5.4. Integración MQTT - Node-RED

La conexión de los flujos de Node-RED con MQTT se realiza a través de unos nodos (figura 4.12) desarrollados por terceros que ya han sido mencionados en el apartado 4.2.3.

El objetivo final de que MQTT esté integrado en Node-RED, es poder controlar toda la red domótica desde un dispositivo central (como una tablet) cuyo software únicamente interactúe con el topic en cuestión.

En el proyecto RoboHealth se han desarrollado dos flujos. Uno de ellos publica los estados de los dispositivos conectados a Vera y Zolertia 2 en el topic *Robohealth/room/devices*; mientras que el otro está suscrito al mismo topic con la intención de actuar de acuerdo a los mensajes codificados que se publican.

Como se explicó en el apartado 4.2.2, cabe recordar que los mensajes para comandar dispositivos siguen el siguiente formato:

$$\{ 'id' : xx, 'atrib1' : yy, 'atrib2' : zz, (...) \}$$

Node-RED se encarga de añadir los atributos y sus valores al mensaje *msg* para actuar en consecuencia. Usando el identificador 99, Robohealth Arm requiere de los atributos **shoulder** y **elbow**.

Así, por ejemplo, si se quiere comandar Robohealth Arm desde MQTT sólo será necesario publicar un mensaje como el siguiente en el topic *Robohealth/room/devices*.

$$\{ 'id' : 99, 'shoulder' : '8C', 'elbow' : '5A' \}$$

5.4.1. Flujos MQTT

En la actual sección se presentan los flujos de Node-RED que interactúan con MQTT, bien publicando en cierto *topic* o subscribiéndose al mismo.

El flujo presentado en la figura 5.6 muestra la etapa de suscripción. Está permanentemente a la escucha de las publicaciones de *Robohealth/room/devices* y transforma el formato del mensaje MQTT en un objeto *msg* propio de Node-RED gracias al nodo *JSON*⁴. El nodo *Switch* sirve para clasificar el mensaje en función de su identificador. A partir de ese punto, la acción es específica del dispositivo representado por el identificador recibido.

En el caso del brazo robótico, con identificador 99, se sitúa su salida en último lugar. La función *Adapt msg* genera un *msg.payload* compatible con el script *sendAT.py* (anexo B.1), que se ejecuta posteriormente.

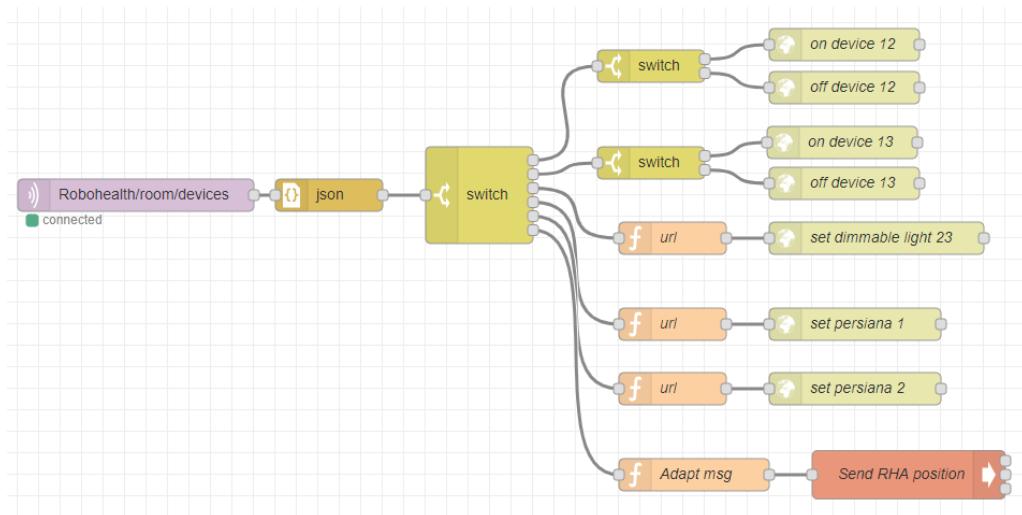


Figura 5.6: Flujo de suscripción a MQTT

En la figura 5.7 se muestra el flujo que publica en MQTT el estado de los dispositivos relacionados con la red Vera o con Zolertia 2 con fines informativos para los dispositivos suscritos al topic.

5.5. Integración Node-Red - XBee

La integración de Node-RED con los módulos Xbee se realiza a través de comunicación serial comandada a través de los distintos scripts recogidos en el Anexo B.

En Node-RED los scripts son llamados usando el nodo Exec (comentado en el apartado 4.2.3). Los nodos de envío se ejecutan rápidamente sin problemas y no

⁴Se trata de un nodo que ejecuta conversiones en ambos sentidos entre un string JSON (el mensaje de MQTT) y un objeto de JavaScript (el objeto *msg* de Node-RED)

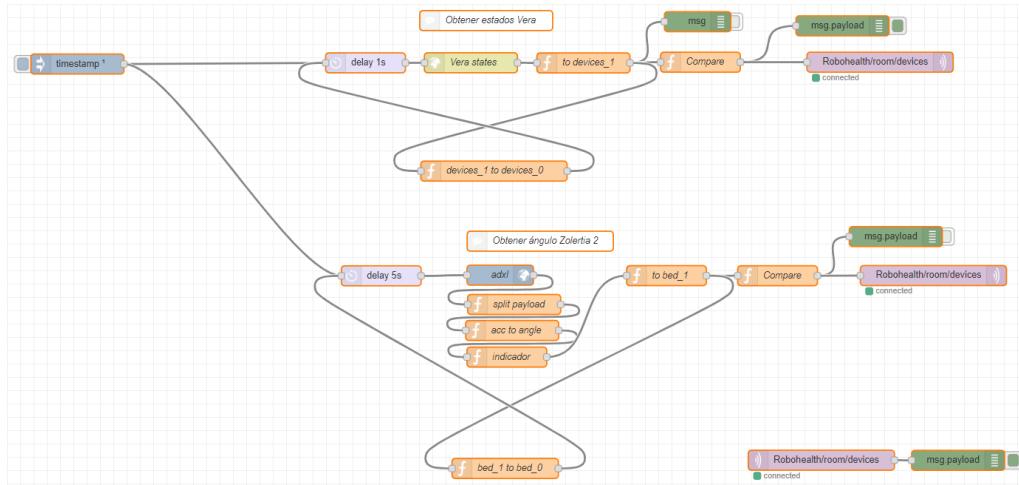


Figura 5.7: Flujo de publicación en MQTT

generan salidas. El nodo de recepción, en cambio, permanece durante un tiempo indefinido volcando mensajes de estado del brazo robótico al flujo de Node-RED. Como inconveniente, comentar que no ha sido posible sacar las salidas correctas del programa por la salida estándar *stdout* en Node-RED. Como se puede observar en el código B.3 y en el flujo de recepción de datos (figura 4.25, este problema se ha solucionado recurriendo a la salida de error *stderr* que, en Node-RED, sí que funcionaba adecuadamente.

La comunicación serial se realiza con la configuración estándar (código 5.2), normalmente usada por defecto para la mayoría de dispositivos. Se establece, eso sí, una tasa de baudios de 115200 que, recordamos, era el máximo admisible por los módulos XBee

Código 5.2: Configuración serial

```

1  ser = serial.Serial(
2      port='/dev/ttyACM0',
3      baudrate = 115200,
4      parity=serial.PARITY_NONE,
5      stopbits=serial.STOPBITS_ONE,
6      bytesize=serial.EIGHTBITS,
7      timeout=1
8  )

```

No es necesario hacer ninguna modificación a la información más allá de los scripts, por lo que la configuración hardware en modo USB de la XBee Shield encaja perfectamente en nuestras necesidades. Como se ha indicado previamente en el apartado 4.3.2, es necesario retirar el microcontrolador del Arduino UNO y la unión funcionará como una conexión directa serie entre la Raspberry Pi y el módulo XBee Coordinador (figura 5.8).

También existe la posibilidad de conectar el módulo Xbee directamente usando los pines dedicados a comunicación serie del módulo y de la Raspberry Pi (figura 5.9). Se ha optado por la primera solución porque esta segunda va a ser la aplicada en el siguiente apartado y por facilitar en montaje y la conexión en el lado de la unidad de mando.



Figura 5.8: Conexión serial XBee-RPi

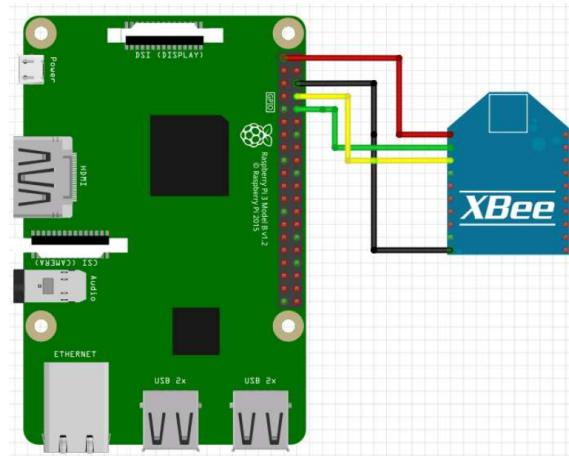


Figura 5.9: Conexión serial alternativa XBee-RPi

5.6. Integración XBee - RHA

Por su parte, no hay otra opción para conectar el módulo XBee al Arduino Mega del RoboHealth Arm que usar los pines seriales. Estos pines son, en el módulo XBee, DOUT y DIN y, en el Arduino Mega, D0(RX) y D1(TX)⁵. El diagrama de conexiones se muestra en la figura 5.10. Existe la posibilidad de usar alguno de los dos puertos seriales restantes⁶ para evitar cualquier potencial interferencia en la comunicación por el USB pero esto implicaría modificar el código cambiando el puerto serial de la interfaz *Pyinterface*.

⁵El Arduino Mega posee otros tres puertos seriales que podrían ser igualmente usados

⁶Uno de los tres está usado por los servos

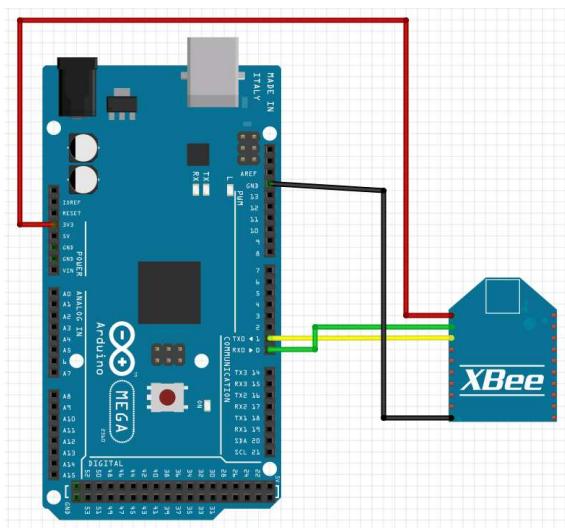


Figura 5.10: Conexión serial Arduino Mega - XBee

Capítulo 6

Resultados y discusión

A continuación, se exponen y analizan los resultados alcanzados tras la realización del proyecto. La inclusión de archivos de log nos ofrece datos del estado del sistema que permiten estudiar la respuesta del brazo robótico ante los comandos.

6.1. Resultados

En términos generales, se han obtenido unos resultados satisfactorios en cuanto al funcionamiento del proyecto en todas sus etapas.

A continuación se detallan los resultados obtenidos a partir de los datos recogidos de archivos de log y pruebas en vídeo. En el Anexo C.2 se recopila documentación en relación a las pruebas realizadas

6.1.1. Test de eficiencia en recepción

Se analiza el log de recepción de información para comprobar la correcta recepción de los datos.

En la figura 6.1 se encuentran cuatro test en los que se representa la posición del hombro en función del tiempo relativo de test. Los frames incorrectamente recibidos se detectan cuando se recibe un 0 como posición articular.

Analizando los datos de los logs, se puede observar que de un total de *2864 frames* recibidos, tan sólo 2 lo hicieron de manera incorrecta. Teniendo en cuenta que RoboHealth Arm está programado para emitir un frame de estado cada 0,5 segundos, cabría esperar que enviara un total de 2984 frames. Esto supone una recepción adecuada del *95,91 %* de los frames. Del porcentaje restante, el *4,02 %* corresponden a mensajes teóricamente enviados por RHA pero que no han sido captados por el módulo XBee y el *0.07 %* son mensajes recibidos que contienen información errónea.

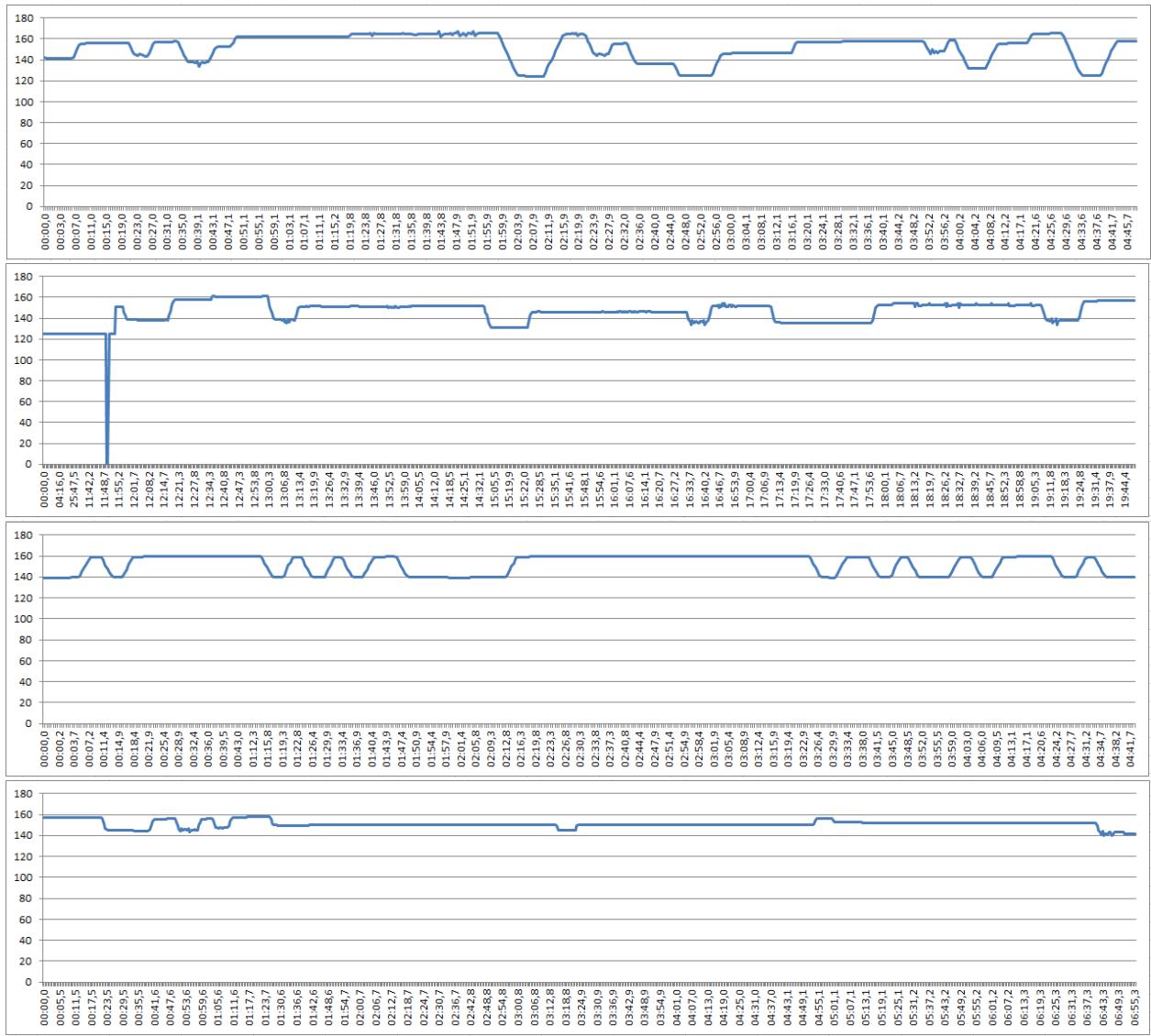


Figura 6.1: Test de recepción de datos

6.1.2. Test de eficiencia en emisión AT

Se analiza una serie de envíos de información en modo AT y se compara con el movimiento del brazo para obtener el porcentaje de éxito en la emisión.

En la figura 6.2 se puede observar la posición de cada articulación con respecto al tiempo. Las líneas rojas verticales corresponden a envíos de información. Analizando la información, podemos concluir que, de 33 envíos, se recibieron correctamente 31. Dos envíos no llegaron a su destino, obteniendo un porcentaje de éxito del 93,93 %.

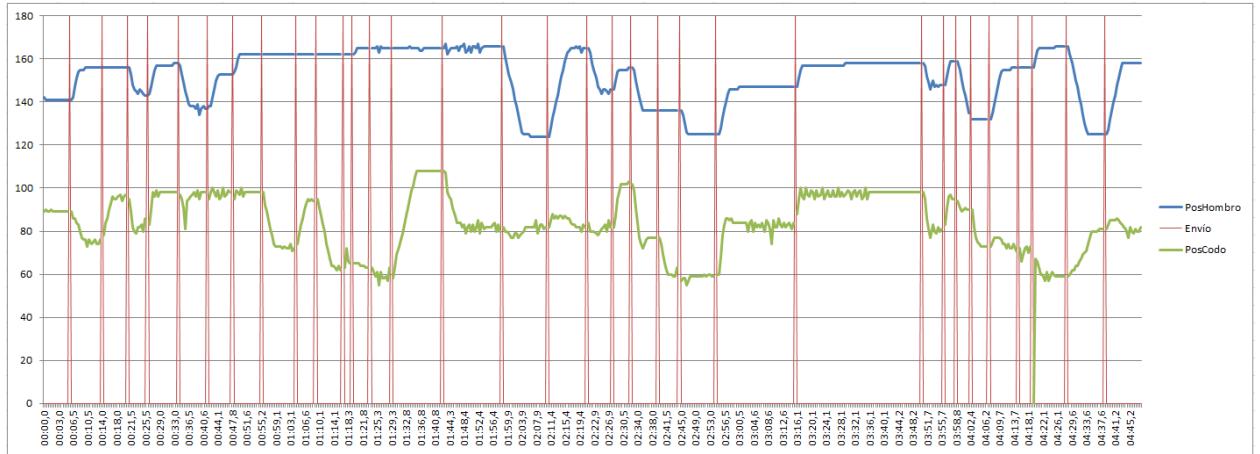


Figura 6.2: Test de emisión AT de datos

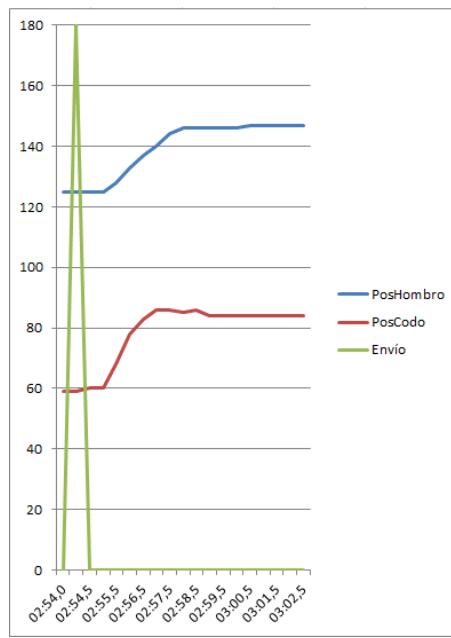


Figura 6.3: Detalle de la respuesta

6.1.3. Test de eficiencia en emisión API

6.1.4. Test de tiempos de respuesta

El objetivo es usar los logs para obtener el valor del tiempo transcurrido entre que se envía un paquete de información y el robot empieza a moverse¹. En la figura 6.3 se puede ver el detalle de una de estas respuestas.

El tiempo de respuesta medio resultante es de *0.73 segundos*. Este valor se obtiene como la media del tiempo de respuesta de una muestra con *31 envíos exitosos* de información aunque hay que tener en cuenta que el periodo de refresco de la in-

¹En la práctica, será el valor del tiempo de recepción del frame de estado en el que se detecta un cambio de posición en las articulaciones, por lo que cabe esperar que el tiempo de respuesta sea algo menor

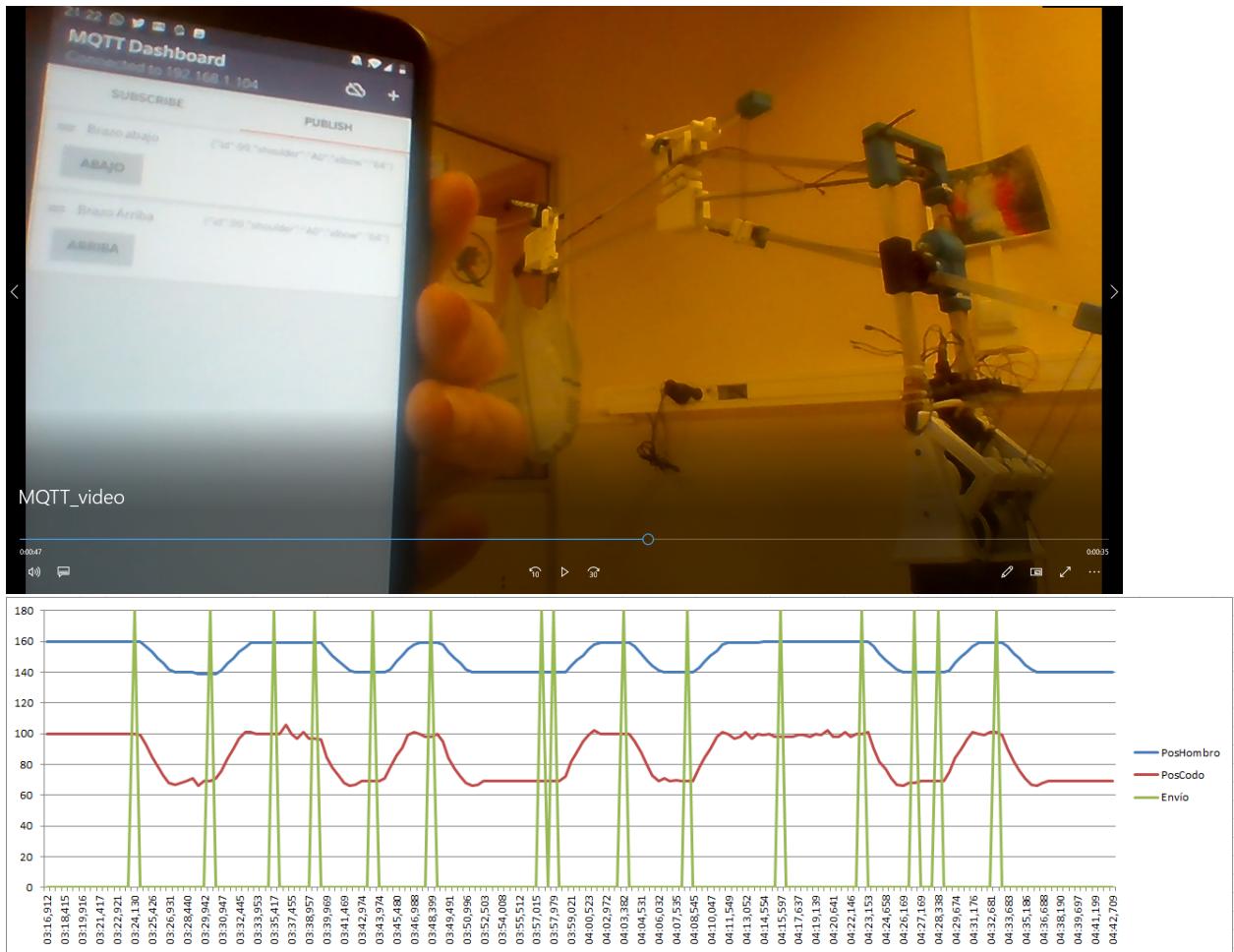


Figura 6.4: Test de uso vía MQTT

formación de las articulaciones que proporciona el RHA es de aproximadamente 0.5 segundos. Se puede comprobar la varianza de los datos que demuestra esta situación en los datos adjuntos en el Anexo C.2. Una varianza representativa² dificulta medir con precisión el tiempo de respuesta en una muestra con datos limitados, como es el caso.

6.1.5. Test de MQTT

A través de pruebas en vídeo sincronizadas con los archivos de log de envío, se trata de obtener un porcentaje de éxito de los comandos desde MQTT.

Con la figura 6.4 uno puede hacerse una idea del proceso seguido. Se aisló un log de tal manera que sus datos coincidieran con la grabación de las acciones realizadas desde un dispositivo conectado a la red MQTT y la respuesta del brazo. Así, contando las acciones indicadas desde MQTT, el número de envíos de información vía XBee y las reacciones del brazo; se puede sacar el rendimiento de la red en cada

²La varianza es representativa en un caso como este cuando el error medio se aproxima al valor medio de las muestras

punto.

Los resultados son los siguientes: 17 veces se mandó al brazo cambiar su posición desde el móvil. De ellas, 15 fueron transmitidas a través de XBee y se provocaron 11 movimientos del brazo.

En porcentajes, el brazo ejecutó el 65 % de las órdenes enviadas desde el móvil, siendo el 88 % de ellas las que se transmitieron vía radiofrecuencia.

6.2. Discusión

Como se ha comentado antes, se han obtenido buenos resultados en general. Se presenta una forma versátil y fiable de comandar el robot.

En cuanto a la recepción, los resultados son excepcionales. Un porcentaje muy alto (95,91 %) de los frames de estado son captados por la red domótica. Estos frames de estado, en el funcionamiento usual de la red, cumplirían la única función de representar en la interfaz la posición articular del hombro y del codo con la tasa de refresco programada en RHA. La pérdida de un frame de estado no es crítica en sí porque la representación mantendría la posición previa hasta recibir un nuevo frame de estado medio segundo después. Lo crítico es evitar la recepción por parte de la red domótica de un frame de estado, identificado como tal, pero con información corrupta, a medias o inexistente. En ese caso, el frame pasaría los filtros de representación y acabaría mostrando una posición errónea. Los datos demuestran que esta situación es residual (0,07 %) así que no supone mayor problema.

La emisión de órdenes en modo AT mantiene un porcentaje de éxito bastante alto (93,93 %) haciendo uso de su sencillez y de ser el modo principal sobre el que está pensado el protocolo de comunicación y, por lo tanto, estar más probado.

En el envío en modo API aumenta la tasa de error a pesar de mantener un porcentaje de éxito bastante sólido. Probablemente sea debido a un frame más largo que requiere de más comprobaciones antes de ser interpretado y a un código de recepción imperfecto. En parte, es el precio a pagar por las ventajas que aporta (explicado en detalle en el apartado 4.3.1).

Los tiempos de respuesta a los envíos de comandos son realmente buenos. A pesar de la dificultad para medirlos de manera precisa, se puede asegurar que no suben del segundo; lo que es más que suficiente para la aplicación descrita.

MQTT introduce una etapa más de transmisión de la información en la que se puede perder. En este caso, las pruebas muestran que los escasos comandos MQTT perdidos se han producido en situaciones en las que se enviaban de manera extremadamente consecutiva. Esto hace que su pérdida no resulte un verdadero problema. Sin embargo, en este test sí que ha aumentado el porcentaje de error en la transmisión de Node-RED al brazo. Hay pocas explicaciones ya que se trata de exactamente el mismo script usado en la sección 6.2, pero podría deberse al tamaño excesivamente

limitado de la muestra grabada en vídeo.

Por último, merece la pena indicar que, aunque no existan pruebas ni datos que lo demuestre, los errores de emisión parece que tienden a producirse cuando el brazo robótico entra en un estado de oscilación. Esta situación es relativamente frecuente y puede observarse en las gráficas previas, como es el caso de la figura 6.4.

Capítulo 7

Gestión del proyecto

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

7.1. Ciclo de vida

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

7.2. Planificación

Se puede indicar mediante un diagrama de Gantt.

7.2.1. Planificación inicial

7.2.2. Planificación final

7.3. Presupuesto

7.3.1. Personal

7.3.2. Material

7.3.3. Resumen de costes

Capítulo 8

Conclusiones

Se presentan a continuación las conclusiones...

8.1. Conclusión

Una vez finalizado el proyecto...

8.2. 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[12]

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

La edición de scripts y, en general, documentos de texto se realiza a través del programa *gedit*. Es posible descargarlo e instalarlo a través del comando

```
1 $ sudo apt-get install gedit
```

Gedit es un editor de texto muy popular en sistemas basados en GNU/Linux, sencillo y ligero. Permite hacer poco más que editar el código con las herramientas más básicas, pero esto será suficiente. Se ejecuta con el comando siguiente, pero indicando el archivo que se quiere editar:

```
1 $ gedit /home/pi/Xbee/receive.py
```

Los archivos con extensión *.py* son scripts de Python que deben ser compilados y ejecutados. Para este cometido, existe un comando con el mismo nombre que el lenguaje. Para descargar Python, existe el siguiente comando²:

```
1 $ sudo apt install python2.7 python-pip
```

Existen versiones posteriores de Python pero los scripts del presente proyecto han sido desarrollados de acuerdo a Python 2.7.13. Para comprobar si Python ya está instalado y su versión, se puede correr el siguiente comando:

```
1 $ python --version
2 Python 2.7.13
```

Por último, a continuación se indica un ejemplo del comando usado para compilar y ejecutar un script en Linux (comando A.3). Se deberá indicar el nombre del archivo que al que queremos aplicar la acción. Si el script requiere de atributos de entrada, se deberán situar a continuación del nombre del archivo.

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

²Se suponen los repositorios actualizados

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-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 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     ser.write(values)
39     logging.info('API_Package_sent')
40
41 else:
42     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     stopbits=serial.STOPBITS_ONE,
16     bytesize=serial.EIGHTBITS,
17

```

```
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. Documentación de XBee Shield

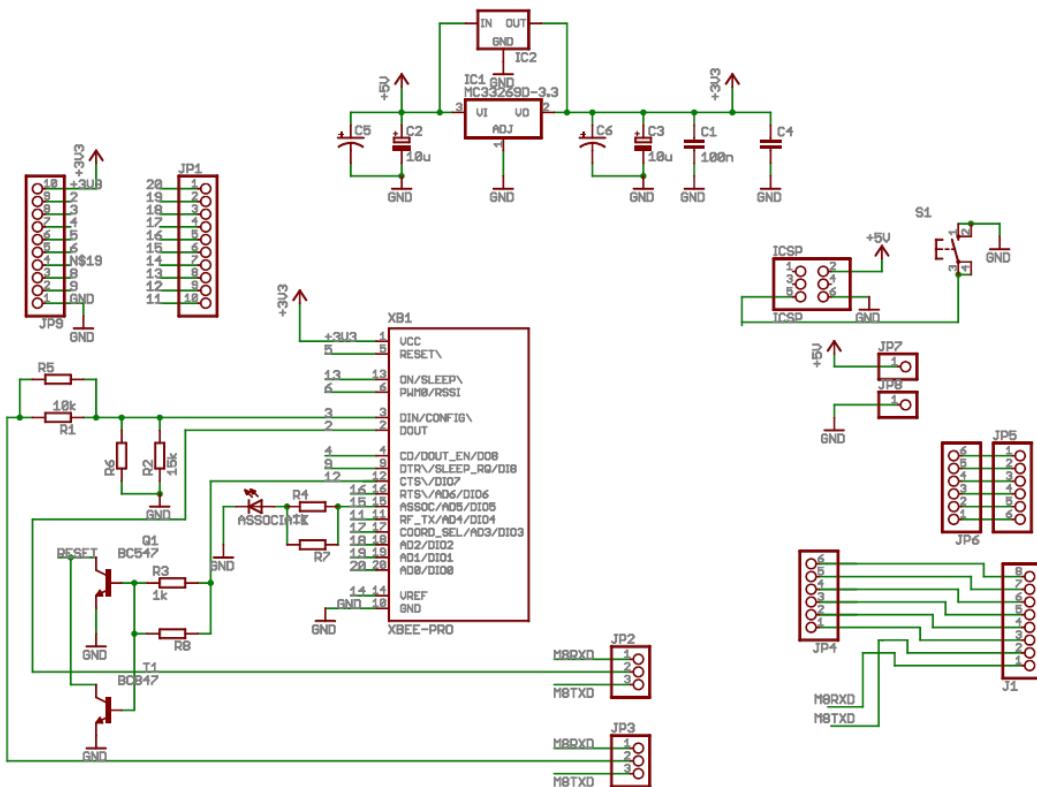


Figura C.1: Esquemático de XBee Shield

C.2. Pruebas del proyecto

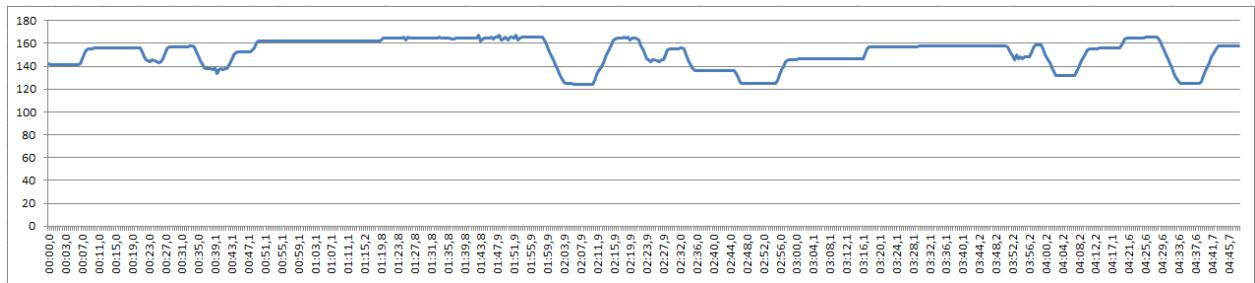


Figura C.2: Test de recepción de datos AT

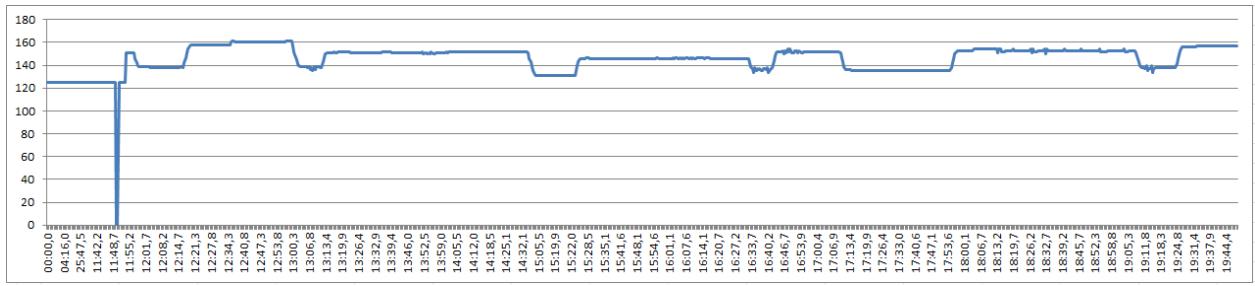


Figura C.3: Test de recepción de datos genérico 1

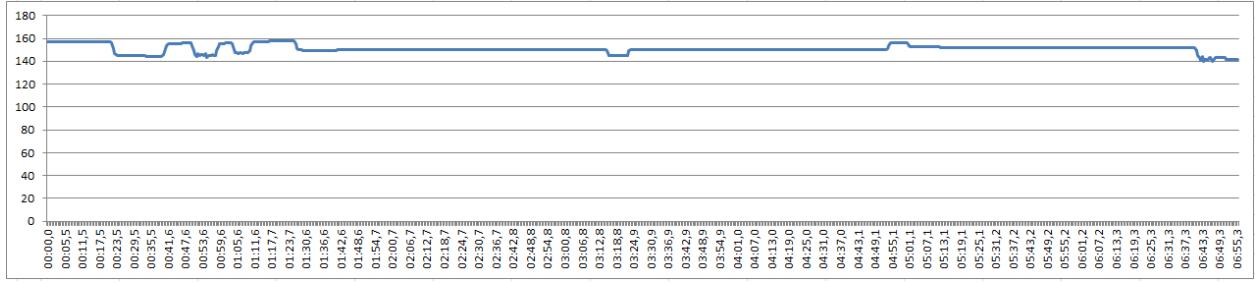


Figura C.4: Test de recepción de datos genérico 2

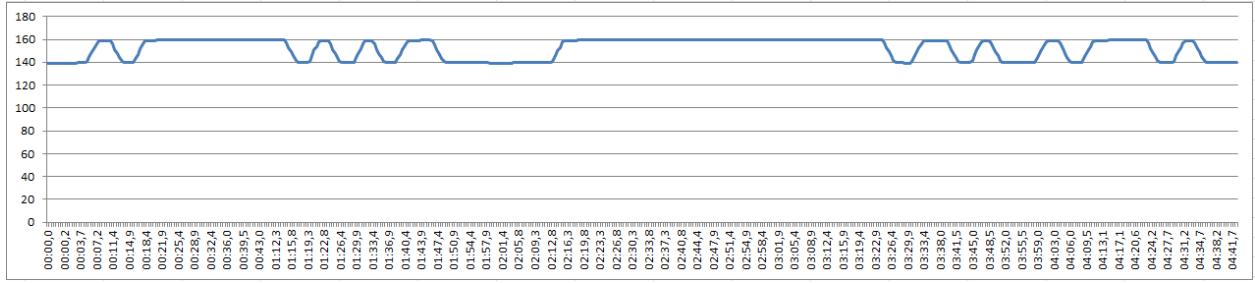


Figura C.5: Test de recepción de datos MQTT

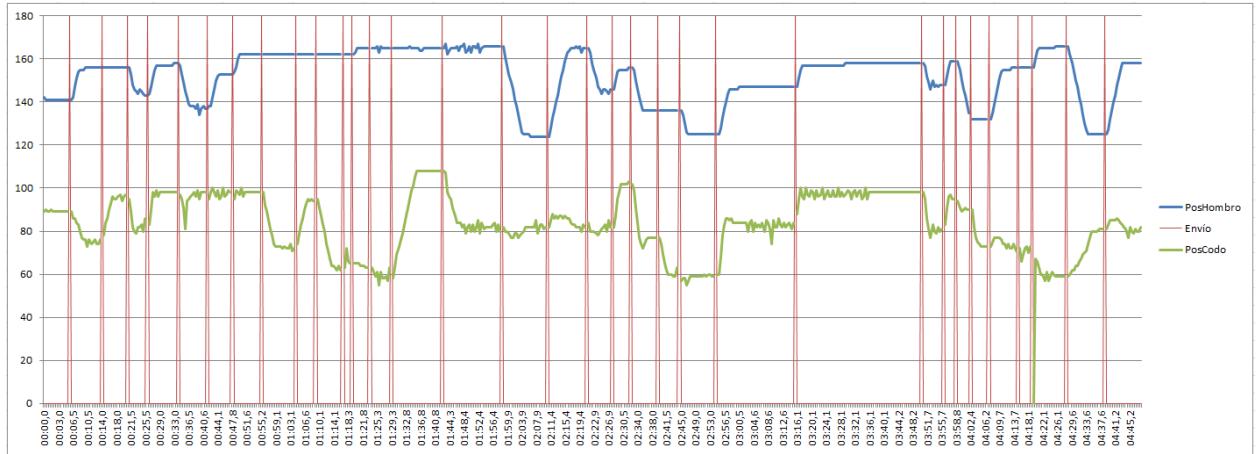


Figura C.6: Test de emisión AT de datos

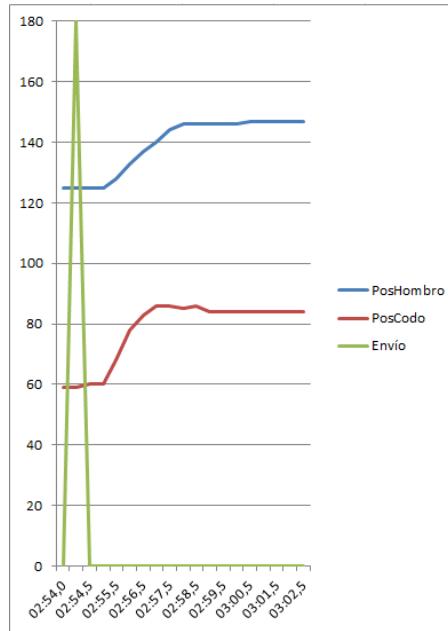


Figura C.7: Detalle de la respuesta ante un paquete de mando

Muestra	tiempo	Muestra	tiempo	Muestra	tiempo	Muestra	tiempo
1	0.767	9	0.295	17	0.667	25	0.686
2	0.48	10	0.852	18	0.885	26	0.908
3	0.822	11	0.896	19	0.702	27	0.76
4	0.261	12	0.909	20	0.658	28	0.962
5	0.487	13	0.524	21	0.576	29	1.001
6	0.544	14	0.524	22	1.037	30	0.895
7	0.793	15	0.931	23	0.482	31	0.889
8	0.942	16	0.925	24	0.648	-	-

Tabla C.1: Estructura del frame de comando

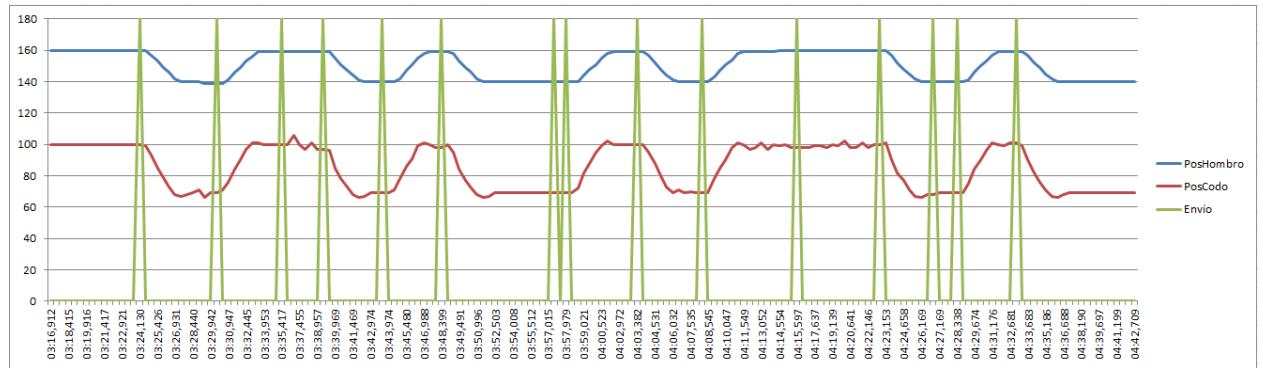


Figura C.8: Test de uso MQTT

Bibliografía

- [1] Ardusmarthome url: <http://openaccess.uoc.edu/webapps/o2/bitstream/10609/64347/8/msanchezmunoz0TFM0617memoria.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] Digi rf documentation pdf url: <https://www.digi.com/resources/documentation/digidocs/pdfs/90001942-13.pdf>.
- [5] Diseño y construcción de un brazo robótico para el apoyo a la interactividad de personas enfermas.
- [6] Dispositivos lógicos programables - la comunicación serie. url: http://perso.wanadoo.es/pictob/comserie.htm#conexion_de_un_microcontrolador_al Puerto_serie_del_pc.
- [7] Github - enheragu/rha url: <https://github.com/enheragu/RHA>.
- [8] Home automation system url: <https://www.digi.com/resources/project-gallery/home-automation-system>.
- [9] Ieee 802.15.4 url: https://es.wikipedia.org/wiki/IEEE_802.15.4.
- [10] Node-red based custom full-room wake-up light url: <https://www.wouterbulten.nl/blog/tech/custom-wake-up-light-with-node-red/>.
- [11] Primeros pasos con mqtt url: <https://ricveal.com/blog/primeros-pasos-mqtt/>.
- [12] Raspberry pi official website url: <https://www.raspberrypi.org/documentation/>.
- [13] Raspberry pi3 + xbee + xbmq + mqtt + node-red iot url: <https://www.instructables.com/id/Raspberry-Pi3-XBee-MQTT-Node-Red-IoT/>.
- [14] 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/>.

- [15] Sistemas de radiocomunicaciones móviles. url: <https://personal.us.es/murillo/docente/radio/documentos/tema9.pdf>.
- [16] 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/>.
- [17] Solar vehicle url: <https://www.digi.com/resources/project-gallery/university-of-minnesota-solar-vehicle> web oficial: <https://umnsvp.org/>.
- [18] Xbee official website url: <https://xbee.cl/>.
- [19] Xbee series 2 oem rf modules datasheet url: <http://www.farnell.com/datasheets/27606.pdf>.
- [20] Zigbee alliance official website url: <https://www.zigbee.org/>.
- [21] Real Academia de Ingeniería de España. *Diccionario Español de Ingeniería*. 1 edition, 2014.
- [22] National Instruments. *LabVIEW Graphical Programming Course*. National Instruments, Elizabeth Gregory, Malan Shiralkar, Harika Basana, 4.6 edition, 2004.
- [23] Cisco Systems. *Fundamentos de Redes Inalámbricas*. Pearson Educación, S.A., 1 edition, 2006.
- [24] Diego M. Zigootto. *Las mil y una curiosidades de Buenos Aires*. Grupo Norma, 2008.