Universidad de Sevilla

Escuela Técnica Superior de Ingeniería

Grado en Ingeniería de Tecnologías Industriales

Intensificación en Automática

Departamento de Ingeniería de Sistemas y Automática

Trabajo de Fin de Grado

DESARROLLO E IMPLEMENTACIÓN DE PROTOCOLO DE COMUNICACIÓN INTER NODO PARA APLICACIONES RANGE ONLY SLAM

AUTOR: CARLOS CORCHADO MIRALLES

TUTORES: ARTURO EUGENIO TORRES GONZÁLEZ

JOSÉ RAMIRO MARTÍNEZ DE DIOS

15 de junio de 2015

Agradecimientos

'*El genio es un uno por ciento de inspiración y un noventa y nueve por ciento de sudor*'.

Thomas A. Edison

Sería imposible dar por finalizada esta etapa de mi vida sin hacer mención a todas aquellas personas que me han ayudado a alcanzar este objetivo tan importante.

Desde muy pequeño he sido una persona realmente cabezona, constante, que cuando algo se propone no para hasta conseguirlo, o al menos lo intenta con todas sus fuerzas. Por suerte, a lo largo de estos cinco intensos años, todas las veces que me he tropezado siempre he tenido a mi familia conmigo, y cada uno de ellos me ha ayudado de la mejor forma que ha podido. Mi padre, un eterno luchador que nunca se da por vencido. Él tira del carro aunque no tenga ruedas, puede con todo y así lo demuestra y me lo enseña cada día. Mi madre, el ejemplo de fortaleza por antonomasia. Ella, y sólo ella, sabe como seguir adelante con una sonrisa en la cara, aunque se encuentre en una de las peores situaciones, reparte felicidad y ayuda a todo el que puede. Mi hermano es el motivo por el que yo comencé esta carrera. Él es el referente, mi mejor ejemplo, una de las personas que más me ha ayudado a esquivar gran parte de los obstáculos que iban surgiendo y me ha ido guiando durante todo este tiempo de la mejor forma que ha podido. Gracias a todos.

Así mismo, mis compañeros y amigos, sin lugar a duda ellos han sido esa vía de escape a la que recurría en los largos días en la escuela, simplemente recordar la cantidad de anécdotas me hace sonreir. Gracias a todos.

Pero sin lugar a duda, tuve la suerte de encontrar al pilar más importante de mi vida, mi chica. Ella ha sabido escucharme, apoyarme y animarme, darme fuerza, quererme. De verdad, gracias.

Por último, quiero agradecer con gran cariño a Arturo y Ramiro, profesores del Departamento de Ingeniería de Sistemas y Automática. Gracias a ellos he tenido el placer de realizar este trabajo y de aprender lo que es la ingeniería realmente, sinceramente gracias por vuestro apoyo.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Objetivo

El principal propósito de este trabajo es desarrollar un protocolo de comunicación inter-nodo para intercambio de medidas de distancia. Todos los nodos formarán parte de una red de sensores y por ello, el primer objetivo será implementar un algoritmo genérico de toma de medidas que permita la adquisición de datos entre los distintos dispositivos.

Se quiere conseguir que una serie de nodos, conectados a una misma red, tengan la capacidad de medir distancias entre ellos según el protocolo vaya ordenando. Concretamente se desea que haya un nodo, conocido como nodo móvil, el cual vaya pidiendo a cada uno de los otros nodos que realicen medidas respecto al resto para posteriormente enviar dichas medidas al nodo móvil. Este nodo móvil tiene como tarea final generar un set de datos con todas las medidas obtenidas.

1.2. Contexto

Una red inalámbrica de sensores consiste en un conjunto de nodos compuestos por sensores, caracterizados por su autonomía energética, disposición para ciertas labores de cómputo y comunicación inalámbrica.

Por su parte, los sensores que componen los nodos de una red pueden tener diferentes características según el tipo de magnitud física que midan, tales como temperatura, presión o, como en el caso de este trabajo, distancia. A lo largo del presente documento se estudiarán las diferentes tecnologías existentes dentro de los sensores de medida de distancia. Dichas tecnologías son principalmente dos. La primera, RSSI, se basa en la medida de la potencia de una señal de radiofrecuencia. En segundo lugar se tiene TOF, basada en el tiempo que tarda una señal de radiofrecuencia en ser transmitida entre dos dispositivos.

Una vez estudiadas las redes inalámbricas de sensores se procederá a realizar la selección de las plataformas que van a constituir el sistema. Tanto el Hardware como el Software serán analizados cuidadosamente para lograr optimizar las características del sistema, que son:

- Escalabilidad: capacidad de ampliar o disminuir el sistema sin realizar modifiaciones significativas.
- Modularidad: posibilidad de trabajar por partes independientes para depurar errores con mayor facilidad.
- Comunicación Asíncrona: comunicación entre nodos de la red de forma ordenada pero no necesariamente síncrona.
- Tiempo de transmisión: Control y acotación del tiempo de medida, unido al de envío y recepción de datos entre los nodos.

En este momento se tratará de realizar la implementación del algoritmo, basado en todos los estudios previos que se irán tratando a lo largo de los capítulos. Finalmente, una vez implementado, se procederán a realizar pruebas. Se comprobará cómo, la automatización del sistema, va a ser algo determinante para lograr una mayor eficiencia, convirtiéndose en una de las características más importantes del sistema.

El desarrollo e implementación de esta herramienta no tiene únicamente caracter teórico. Realmente, las aplicaciones que posee son tremendamente amplias y en su gran mayoría están relacionadas con lo que se conoce como RO-SLAM, técnica derivada de uno de los problemas más importantes de la robótica que permite darle autonomía completa a vehículos móviles, denominado SLAM.

El SLAM (*Simultaneous Localization And Mapping*), o en español, Localización y Generación de Mapas Simultáneos, es una técnica utilizada por robots y vehículos autónomos. Consiste en un proceso mediante el cual un robot móvil puede construir un mapa de su entorno, desconocido a priori, y localizarse adecuadamente dentro de este mapa en tiempo real.

Más tarde surge el RO-SLAM (*Range Only* SLAM) como combinación de las redes de sensores y robots móviles, que ahora cooperan entre ellos. Esta nueva técnica es SLAM usando únicamente medidas de distancia.

Partiendo de estudios previos [1], se llega a la conclusión de que los sensores no tienen por qué considerarse elementos pasivos dentro de la red inalámbrica de sensores, sino que pueden intervenir beneficiosamente en la toma de medidas, computación y comunicación.

En este caso, las medidas que se toman son únicamente de distancia, teniendo como característica principal la gran ventaja de no precisar una línea directa de visión entre cada par de sensores para tomar la medida, al contrario de lo que ocurre con el SLAM puro donde se precisan cámaras u otro tipo de sensores donde la visualización del medio es esencial.

Dicho esto, se puede observar que el desarrollo y la implementación del algoritmo que se presenta en este proyecto puede ser de gran ayuda para este campo de la robótica, permitiendo realizar futuros trabajos de experimentación.

Capítulo 2

Estado del Arte

2.1. Redes de sensores

Una red inalámbrica de sensores o WSN (*Wireless Sensor Network*) consiste en un conjunto de nodos que poseen distintos sensores (típicamente de temperatura, humedad, luminosidad,...), caracterizados por su autonomía energética, disposición para ciertas labores de cómputo y comunicación inalámbrica.

El interés por este tipo de sistemas distribuidos en la industria se está viendo incrementando cada vez más debido a las grandes cualidades que proporciona: capacidad de despliegue rápido, posibilidad de ubicación variable y bajo coste. Si a todo esto se le añade el reducido tamaño de los dispositivos y la posibilidad de escalado, dicho interés aumenta aún más ya que la variedad de campos de aplicación se hace realmente amplia.

Cada uno de los nodos de la red se compone de sensores, transmisor/receptor y un dispositivo con microcontrolador. El objetivo es hacer que un cierto número de estos nodos formen parte de un mismo sistema conocido como red de sensores. Si bien es cierto que, aunque la capacidad de procesamiento de datos de estos dispositivos no es muy elevada debido a limitaciones técnicas, al combinar la información obtenida por cada uno de los nodos los resultados de las mediciones de una cierta cualidad del medio físico en el que se sitúan pueden llegar a tener gran detalle.

Llegados a este punto, se podría describir una red de sensores como un grupo de nodos que se coordinan para llevar a cabo una aplicación específica. Al contrario que en las redes tradicionales, actualmente las redes de sensores realizan sus tareas de manera más o menos precisa en función de la densidad del despliegue y la correcta coordinación.

Las redes de sensores han evolucionado considerablemente; de estar inicialmente formadas por un número pequeño de nodos conectados por cable a una estación central de procesado de datos, se ha llegado hoy en día a lograr redes de sensores distribuidas e inalámbricas con cientos

de nodos. La necesidad de dotar de estas dos propiedades a las redes de sensores es casi obligada ya que se pueden plantear problemas tales como que la localización de un fenómeno físico sea desconocida. Con este nuevo modelo se permite que los nodos estén considerablemente más cerca del suceso de lo que estaría un único sensor. Además, en muchos casos, el número de sensores necesarios es elevado ya que existe la posibilidad de que hayan obstáculos físicos que obstruyan o distorsionen la línea de comunicación.

Hay que tener en cuenta que el medio del cual se desean extraer las cualidades físicas no tiene por qué poseer una infraestructura que permita el suministro energético, ni la comunicación. Esto puede suceder, por ejemplo, en labores de rescate en montaña donde es evidente la carencia de fuente de energía y comunicativa. Es por ello que, necesariamente, los nodos deben funcionar con pequeñas fuentes de energía, y la comunicación debe ser realizada por medios inalámbricos.

Como se dijo anteriormente, otro de los factores que iba a cambiar en las WSN era que iban a tener capacidad de procesamiento distribuido. El motivo de esto es que, al ser la comunicación el consumidor principal de energía, en un sistema distribuido los diferentes nodos, se necesitarían comunicar desde diferentes distancias. A mayor lejanía mayor consumo. La solución que se dio a este problema fue procesar localmente la información reduciendo así el número de datos transmitidos y con ello disminuyendo el consumo energético. Algunas de las aplicaciones de las redes de sensores son:

- Eficiencia energética (control del uso de la electricidad)
- Entornos de alta seguridad (tales como centrales nucleares o aeropuertos)
- Monitorización del medioambiente y hábitats naturales
- Labores de mantenimiento (temperatura o luz)
- Detección de incendios forestales, terremotos o inundaciones
- Domótica (sensorización de edificios inteligentes)
- Medicina (la reducción del tamaño de los nodos está permitiendo el desarrollo de este campo
- Control de tráfico, etc.

Ni que decir tiene que el potencial que poseen estos sistemas es enorme pudiendo combinarse con otros campos de la robótica. Concretamente existe una nueva vertiente de investigación donde se trata de conseguir la cooperación entre redes de sensores y robots móviles.

Hasta hace relativamente poco no se concebía la idea de la colaboración entre robots, y mucho menos entre robots y redes de sensores [2]. Sin embargo, casi por necesidad, nuevas líneas de investigación tratan acerca de ello ya que van surgiendo aplicaciones donde la cooperación

es intrínsecamente necesaria. El motivo principal es que dichas aplicaciones involucren entornos dinámicos con condiciones cambiantes para la percepción, por ejemplo. En gran parte de estas situaciones, un único robot (dotado de sensores locales como una cámara) no permite conseguir la eficacia y robustez necesaria. Esta serie de nuevas aplicaciones son tales como detección de incendios o robótica de servicios.

Los robots móviles en entornos urbanos pueden ser una idea un tanto futurista para muchos pero es una realidad incipiente en la sociedad. Ya se pueden observar en museos la utilización de robots guía. Para un correcto funcionamiento de estos robots, conocer la ubicación del visitante es primordial ya que la seguridad es muy importante. La incorporación de redes de sensores en museos para este tipo de visitas es necesaria para que la posición del robot y de la persona sea precisa.

Por ello, llegamos a una de las aplicaciones más importantes de las redes de sensores. Conocer la ubicación de un nodo móvil (visitante que carga con un nodo móvil) en una red de nodos fijos a partir de las posiciones de estos para que el robot guía no interfiera en la trayectoria de la persona. Ya que el mapa del entorno es conocido previamente, el problema se centra únicamente en la localización del nodo móvil.

Otro ejemplo podría ser el de tareas de exploración de terrenos desconocidos. Diferentes nodos fijos pueden ser desplegados de forma aleatoria en un determinado emplazamiento donde un robot móvil (considerado nodo móvil) trata de captar toda la información que dicho lugar le proporciona, tales como temperatura, presión atmosférica, humedad, etc. Además, llegados a este punto, la combinación entre la red y el robot va a permitir dotar a dicho robot de autonomía total ya que podrá generar un mapa del entorno a partir de las localizaciones de los nodos fijos y así mismo localizarse en dicho mapa. Esto se conoce como RO-SLAM, concepto que se tratará con detenimiento en puntos posteriores.

2.2. Sensores de distancia

Como se dijo anteriormente, las redes de sensores eran sistemas distribuidos formados por un conjunto de nodos. Cada uno de estos nodos constaba de un sensor, entre otros componentes.

Una de las aplicaciones más directas de las redes de sensores es aquella relacionada con la estimación de la localización de un robot o persona dentro de una red. La forma de enfocar este problema tuvo dos vertientes. La primera de ellas [3] consistía en el empleo de dos nodos únicamente, los cuales detectaban las localizaciones con respecto al objetivo continuamente para finalmente realizar la media de las coordenadas pudiendo localizar el objetivo. Mediante este método, la precisión y resolución de la estimación de la localización se veía afectada directamente por la densidad de los nodos que componian la red en ese area en concreto. Por otro lado se realizaron experimentos como el de [4] donde se partía de tres o más nodos para localizar el

objetivo en una determinada área. En vez de simplemente detección, las distancias entre el objetivo y los sensores era medida en este segundo caso. A la determinación de distancia entre dos elementos es conocida como *ranging*. Usando las medidas tomadas, la localización exacta del objetivo podía ser obtenida mediante técnicas de angulación o trilateración. Por tanto se llega a la conclusión que la densidad de nodos en una determinada area realmente no incrementa la precisión de la estimación de la localización, sino que mas bien depende de la precisión del método de *ranging*.

Para poder lograr obtener información fiable, el estudio de las diferentes tecnologías de medidas de distancia aplicadas a sensores es fundamental ya que van a permitir monitorizar el entorno de manera eficaz.

Hoy en día, existen varias técnicas y tecnologías disponibles para el desarrollo de sistemas distribuidos [5]. Los diferentes requerimientos de cada sistema influencian directamente en la selección de un tipo u otro, dependiendo de la precisión, entorno interior o exterior, técnicas de posicionamiento, método de medida (*ranging*), seguridad y privacidad, restricciones de la WSN, entre otras. Desde el punto de vista de la tecnología, se pueden clasificar en la siguiente figura 2.1.

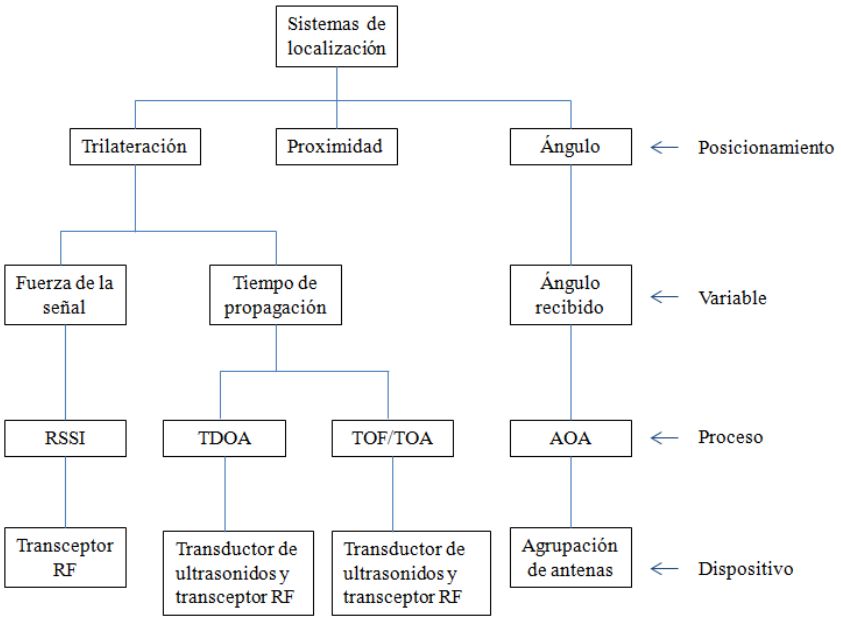

Figura 2.1: Clasificación de los sistemas de localización de seguimiento [6].

2.2.1. RSSI

RSSI (*Received Signal Strength Indicator*) es un indicador de la potencia de señal recibida. Consta de una escala de referencia para medir la potencia a la que una señal de radio es recibida por la antena de un dispositivo en una red inalámbrica.

Existen sensores de distancia que se benefician de este indicador. Obtienen una medida en *dBm* (decibelio-milivatio) de la potencia de la señal de radio recibida. Si bien es cierto, los sensores no están diseñados para realizar este tipo de conversiones RSSI a distancias. Dicha medida en decibelios es transformada, mediante un modelo experimental, en una estimación de distancia absoluta o relativa, según sea el caso, entre dos nodos de la red.

Cabe mencionar que el indicador proporciona la intensidad de la señal recibida, no la calidad de la misma. Es aquí cuando surgen los problemas, ya que la forma a través la cual una onda es propagada en el medio aéreo contiene numerosas distorsiones asociadas a interferencias, además de a las no linealidades propias del modelo de propagación. Los problemas asociados a la propagación se caracterizan por la emisión multitrayectoria. Puede que la señal rebote en algún elemento del entorno y luego llegue al objetivo, dando una medida errónea. Así mismo, las señales transmitidas se ven atenuadas constantemente por el entorno, lo que añade error en la medición.

Es por ello que el hecho de convertir la medida RSSI a distancia no es una tarea trivial. Dicho modelo debería proporcionar la transformación de manera precisa, con posibilidad de ajuste al entorno y capacidad de reducción de errores. Actualmente existen tres modelos de propagación de señal RSSI para redes de sensores inalámbricas [7], *free space model*, *2-ray ground model* y *log-normal shadowing model* (LNSM). Los dos primeros necesitan una serie de premisas para su aplicación según el entorno donde se encuentre la red de sensores, mientras que el último mencionado es un modelo algo más genérico y adaptable.

Sin embargo, a pesar de encontrar esta serie de problemas, los métodos RSSI son los más usados debido a su simplicidad. Cuando anteriormente se veía que la propagación iba a ser un problema, también se considera como una ventaja ya que no son sensores direccionales como los ultrasonidos, lo que amplifica el campo de medida. Por otro lado, el uso de este tipo de tecnología puede ser tanto en interiores como en exteriores, complicándose en interiores debido a que todos los problemas presentados hasta el momento se ven amplificados, tales como el de la propagación multitrayectoria.

Finalmente cabe destacar que, a diferencia de otros métodos donde es necesaria la existencia de protocolos de comunicación para el envío de medidas de distancia, en los sensores dotados de tecnología RSSI, el hecho de realizar la toma de medida no requiere tareas adicionales ya que el valor de la potencia de la señal se mide directamente mediante los mensajes intercambiados entre nodos.

2.2.2. TOF/TOA

TOF (*Time Of Flight*), también conocido como TOA (*Time Of Arrival*) describe una gran variedad de métodos de medida del tiempo que tarda un objeto, partícula u onda de cualquier tipo en viajar una distancia a través de un medio. Esta medida puede ser usada como un estándar

de tiempo, como forma de medida de velocidad o trayectoria en un medio dado. También puede ser empleada para el estudio de una partícula o medio. Las aplicaciones de esta método son incluso mayores que en el caso anterior y pueden ser tales como:

- En electrónica, estimación de la movilidad de los electrones.
- GPS.
- Espectroscopia de infrarrojo, tecnología usada en láseres de medida de distancia.
- Medidor de flujo ultrasonido.

Uno de los ejemplos más conocidos de aplicación del *time-of-flight* antes mencionado es en los láseres. Este tipo de instrumentos son capaces de medir el tiempo que tarda un pulso de luz en viajar al objetivo y volver al origen. Con la velocidad de la luz conocida, se puede obtener una medida precisa del tiempo empleado. Normalmente son múltiples pulsos los enviados para realizar la media de las distancias.

Una señal RF viaja a la velocidad de la luz, y un metro de precisión implica unos 3 nanosegundos de resolución temporal. En un sistema bajo coste y de baja autonomía energética, es dificil conseguir esta precisión. RF en TOF se usa en la tecnología *GPS* (*global position system*) con mucho éxito pero las aplicaciones en sistemas terrestres se han visto limitadas debido a problemas con la resoución temporal, efectos multitrayectoria y coste y complejidad de los sistemas.

Por tanto, la precisión y calidad de las medidas TOF queda delimitada principalmente por sincronización de reloj, ruido, efectos multitrayectoria y errores de muestreo. Así mismo, es necesario conocer bien el medio de propagación, completando por tanto los elementos para determinar la distancia de manera efectiva. Sin embargo, la mayoría de estos problemas están siendo solucionados debido a nuevas tecnologías, como ocurre con el empleo de *CMS* (*Code Modulus Synchronization*) método usado para mitigar los efectos de ruido, sincronización de reloj y errores de muestreo [8].

Es evidente que, al igual que en RSSI, las mediciones TOF están sujetas a efectos negativos procedentes del medio y de errores instrumentales como ocurre con cualquier tecnología inalámbrica. Sin embargo la precisión que se puede lograr en comparación con recibir la intensidad de la señal, es significativamente mayor incluso en las mejores condiciones [9].

Como se verá posteriormente, los sensores empleados para el desarrollo de este proyecto son de tipo TOF y se caracterizan por medir el tiempo que tarda una señal RF (Radio Frecuencia) en ser recibida entre los nodos que forman la red.

2.2.3. Otros

Pese a ser RSSI y TOF las tecnologías más empleadas para calcular medidas de distancia, existen otros métodos algo menos conocidos tales como TDOA (*Time Difference Of Arrival*), FDOA (*Frequency Difference Of Arrival*) y PRT (*Pulse Ranging Technology*).

- TDOA: Al igual que en TOF, la variable a medir va a ser el tiempo de propagación aunque con una diferencia sutil. Mientras que en TOF se calculaba el tiempo de llegada absoluto entre dos estaciones, ahora se calculan las diferencias de tiempo entre la salida de la señal y la llegada a la otra estación. Existen aplicaciones muy conocidas como la localización de teléfonos móviles.
- FDOA: Es una técnica análoga a TDOA para la estimación de la localización de un emisor de radiofrecuencias basada en la observación de otros puntos. A veces ambas técnicas son usadas conjuntamente para mejorar la precisión, consiguiendo geolocalización instantánea en dos dimensiones. La diferencia fundamental es que, en FDOA, los puntos de observación deben estar en movimiento relativo con respecto a los propios puntos y al emisor de la señal, permitiendo estimar la posición del emisor a partir del vector de velocidades generado y de los desplazamientos *Doppler* relativos observados entre pares de puntos.
- PRT: Esta nueva tecnología emplea sensores fotoeléctricos en sus aplicaciones dando una precisión realmente buena, del orden de milímetros tanto a distancias cortas como largas. El principio que usa PRT es similar al TOF, existiendo aplicaciones como láseres.

Capítulo 3

Plataformas empleadas

3.1. Introducción

Una de las partes más importantes de este proyecto son las plataformas que se van a utilizar ya que, gracias a ellas, el sistema funcionará de forma óptima. Si recordamos, el objetivo no era solo crear el protocolo de comunicación dentro de una red de sensores que permitiera el envío y recepción de medidas de distancia entre nodos, sino también crear dicha red.

Es por ello que, para lograr una red sincronizada, eficiente y sin carencias comunicativas, tanto el algoritmo creado como el hardware elegido deben permitir solventar dichos problemas. A lo largo de este capítulo se verán de manera desacoplada las diferentes herramientas que van a intervenir en el proyecto, tanto de hardware como de software.

Por su parte, el hardware implica la creación de los nodos y, por tanto, conlleva directamente la necesidad de encontrar sensores de medida de distancia así como una plataforma que permita dar la conectividad necesaria para crear la red de sensores. En cuanto a software se refiere, la tarea principal es encontrar la forma de crear el protocolo de comunicación entre los nodos.

Este análisis de las herramientas va permitir tener una visión genérica sobre los elementos de los cuales se parten, observando sus ventajas y limitaciones, dando la oportunidad de llegar a un planteamiento del problema que de la opción a desarrollar diferentes soluciones aprovechando al máximo las posibilidades del sistema como conjunto. Finalmente, gracias al estudio de las diferentes plataformas empleadas se comentará la arquitectura tomada del sistema.

3.2. Hardware

3.2.1. Raspberry Pi

Introducción

Anteriormente se realizaba la pregunta sobre cómo se iba a crear la red de sensores. Para ello se comentó que dichas redes se componían de sensores conectados a pequeñas computadoras denominadas nodos.

Para este proyecto se va disponer de las conocidas *Raspberry Pi* (ver figura 3.1). Estos dispositivos son ordenadores de placa reducida de bajo coste, desarrollados en Reino Unido por la Fundación *Raspberry Pi*, teniendo como objetivo principal promover la enseñanza de ciencias de la computación en las escuelas.

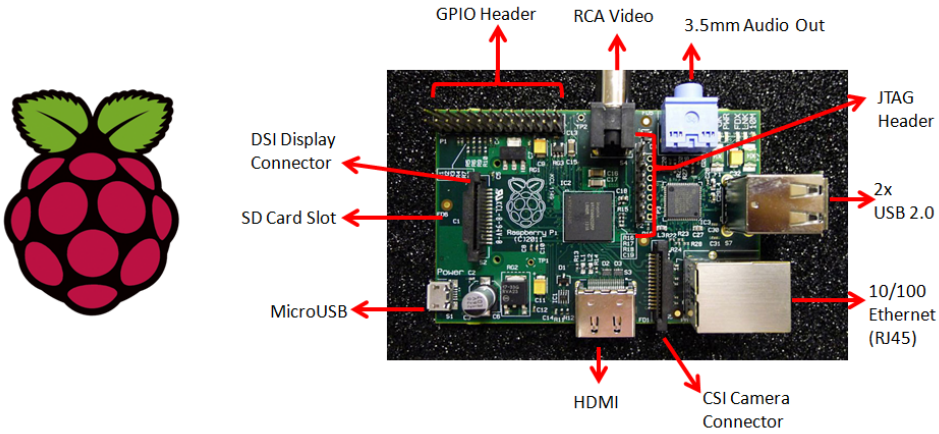

Figura 3.1: A la izquierda Logo de la Fundación *Raspberry Pi*, a la derecha modelo B de *Raspberry Pi*

Existen diferentes modelos pero, en este trabajo, se va a emplear el modelo B. Este diseño se compone de un *System-on-a-chip Broadcom* BCM2835 que consta de un procesador central (CPU) ARM 1176JZF-S a una velocidad de 700 MHz, procesador gráfico (GPU) y memoria RAM 512 MiB. Por otro lado, para el almacenamiento permanente emplea tarjeta SD ya que no incluye disco duro ni unidad de estado sólido; a su vez, no incorpora ni carcasa ni fuente de alimentación. Más adelante se explicarán los detalles técnicos más relevantes.

Estos dispositivos soportan diferentes distribuciones del sistema operativo *Linux* (arquitectura ARM), tales como *Raspbian* (derivada de *Debian*), *RISC OS*, *Arch Linux* y *Pidora* (derivada de *Fedora*).

El sistema operativo utilizado será una modificación de *Raspbian* la cual ya incluye el software que se empleará para la implementación del algoritmo.

Hardware

Como se dijo anteriormente, a continuación se va a proceder a mostrar en una pequeña tabla las características del modelo B [11] empleado para la elaboración de este proyecto, concretamente para la creación de la red de sensores.

Características *Raspberry Pi* Modelo B	
SoC	Broadcom BCM2835 (CPU+GPU+DSP+SDRAM+puerto USB)
CPU	ARM 1176JZF-S a 700MHz
GPU	Broadcom VideoCore IV, OpenGL ES 2.0, MPEG-2 y VC-1, 1080p30H.264/MPEG-4 AVC
Memoria (SDRAM)	512 MiB (compartidos con la GPU)
Puertos USB 2.0	2 (vía hub USB integrado)
Entradas de vídeo	Conector MIPI CSI
Salidas de vídeo	Conector RCA, HDMI,Interfaz DSI para panel LCD
Salidas de audio	Conector de 3.5 mm, HDMI
Almacenamiento integrado	SD/MMC/SDIO
Conectividad de red	10/100 Ethernet (RJ-45) via hub USB
Periféricos de bajo nivel	8 x GPIO, SPI, I2C, UART
Reloj en tiempo real	Ninguno
Consumo energético	700 mA, (3.5 W)
Fuente de alimentación	5 V vía Micro USB o GPIO header
Dimensiones	85.60mm x 53.98mm
Sistemas operativos soportados	GNU/Linux: Debian, Fedora, Arch Linux, Slackware Linux. RISC OS2

Análisis del dispositivo

Una de las características más importantes que beneficia en gran medida el uso de este tipo de terminales es el hecho de tener varias opciones para acceder a ellos y poder realizar modificaciones. Por un lado, y de tanta utilidad para el proyecto, al disponer de entrada HDMI y soporte gráfico tiene la posibilidad de conectar un monitor para realizar los cambios necesarios. Para el caso en cuestión, esto puede verse más como una limitación que una ventaja ya que conectar cada uno de los sensores a una *Raspberry Pi* y estas, a su vez, a diferentes monitores, implica un gasto económico innecesario además de un coste energético muy importante.

Dicho coste energético mencionado no es asumible ya que se va a disponer de baterías para alimentar cada par sensor-computadora por respectivos puertos miniUSB y microUSB. Las

baterías a utilizar serán de 10.000mAh de capacidad y transportan la energía a 5 V y 1 A.

Como alternativa de método de conexión, estos dispositivos al tener la posibilidad de conectarse a una red, disponen de SSH (*Secure SHell*), protocolo que permite acceder a las diferentes máquinas remotas a través de una red mediante un intérprete de comandos. Además de establecer la conexión a dispositivos, este protocolo permite copiar datos de forma segura. Para poder conectar cada computadora a una misma red se emplearán adaptadores WIFI acoplados a uno de los dos puertos USB que la *Raspberry Pi* dispone, y se configurarán para que se conecten a una red local previamente creada.

3.2.2. Sensores ToF: *Nanotron nanoPAN 5375*

Introducción

Los sensores van a ser uno de los componentes de hardware más importantes que se van a emplear en este proyecto. Esto es debido a una simple razón; para conseguir desarrollar un algoritmo basado en RO-SLAM, los nodos deben adquirir un carácter activo en cuanto a lo que a toma de medidas se refiere como se comentó en el capítulo anterior. Para afrontar este proyecto se va a disponer de sensores *Nanotron NanoPAN* 5375.

Nanotron es uno de los proveedores más importantes de productos inalámbricos empleados para ayudar a proteger y buscar personas, animales o bienes valiosos. *Nanotron NanoPAN* 5375 proporciona información de posición realmente fiable teniendo además una eficiencia energética más que correcta.

Se comentó en el capítulo anterior que existían diferentes tecnologías para medir distancias, concretamente para este proyecto se dispone de sensores de radio frecuencia basados en un chip transceptor integrado con funciones TOF capaces de determinar la distancia relativa entre diferentes sensores.

Hardware

El *Nanotron NanoPAN* 5375 es un dispositivo que utiliza un diseño RF ideal para desarrollar productos basados en la tecnología CSS (*Chirp Spread Spectrum*). CSS es una técnica de 'espectro ensanchado' (técnica de modulación empleada para la transmisión de datos digitales y por radiofrecuencia) que utiliza pulsos de frecuencia modulada *chirp* lineales de banda ancha para la codificar información. Siendo un *chirp* una señal sinusoidal cuya frecuencia se incrementa o decrementa a lo largo del tiempo.

Este módulo RF incluye el transceptor *nanoLOC TRX IC* y diferente circuitería necesaria, además dispone de un amplificador de potencia de +20 dBm, así como filtro de paso banda y reloj de cristal.

Concretamente se tiene [12]:

Características específicas de *nanoPAN 5375 RF Module*	
Data rates	250 kbps, 1 Mbps
Ambient temperature range	-40 to +70 C
Supply voltage	2.5 ± 0.2 V
TX current	typ. 210 mA
HC Ranging	(80 MHz, 1 Mbps)
RX sensitivity	typ. -89 dBm
RX current	typ. 51 mA
TX output power	typ. +20 dBm
TX power range	typ. 35 dB
LD Ranging	(80 MHz, 250 kbps)
RX sensitivity	typ. -95 dBm
RX current	typ. 69 mA
TX output power	typ. +20 dBm
TX power range	typ. 35 dB
R Comm	(22 MHz, 250 kbps)
RX sensitivity	typ. -96 dBm
RX current	typ. 34 mA
TX output power	typ. +20 dBm
TX power range	typ. 35 dB
Antenna load impedance	nom. 50 Ohm
SPI clock frequency	up to 24 Mbps

Como se puede observar en estudios previos [10], se puede obtener experimentalmente una caracterización de estos sensores llegando a obtener que el error sigue una distribución normal con media cero y desviación típica $\sigma_m = 0{,}8m$(ver figura 3.2)

Análisis del dispositivo

Como se ha podido observar, este tipo de dispositivos son realmente óptimos y ofrecen una serie de características muy interesantes que va a permitir realizar una toma de medidas con un error muy pequeño. El hecho de estar dotados con comunicación inalámbrica y la robustez en su diseño proporciona un abanico de posibilidades que se intentará aprovechar al máximo en la implementación.

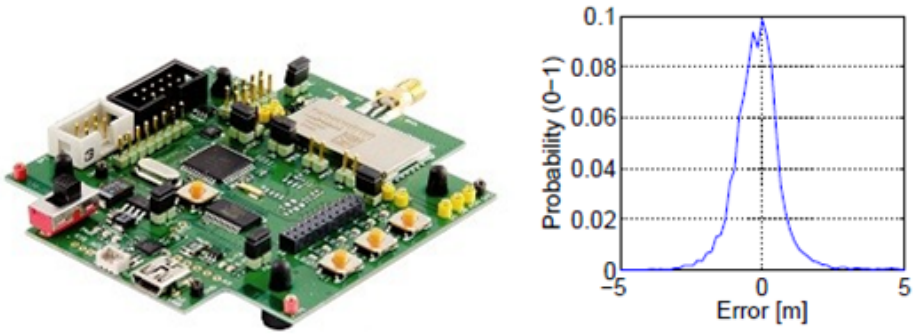

Figura 3.2: A la izquierda sensor *Nanotron NanoPAN* 5375, a la derecha curva de la distribución normal seguida

3.2.3. Hardware de conectividad

Otra de las preguntas a plantearse es acerca de la creación de una red que permita establecer la comunicación entre los nodos, como se comentó en puntos anteriores. Para ello se va a emplear un dispositivo *router WiFi*, concretamente *ASUS DSL-N12E* junto con adaptadores WiFi acoplados a uno de los puertos USB que posee la *Raspberry Pi* (ver figura 3.3).

Figura 3.3: A la izquierda dispositivo router *ASUS DSL-N12E* 5375, a la derecha adaptador WiFi.

Esta será la forma más sencilla de conectar todos los computadores a través de la creación de enlaces con diferentes direcciones IP. La utilización de este elemento proporciona gran versatilidad y facilidad a la hora de realizar envío de paquetes de datos ente máquinas. Sin embargo, van a ser más relevantes las limitaciones existentes. Entre ellas cabe destacar que el número de dispositivos conectados a una misma red va a estar acotado en función del *router* utilizado que, si se emplease uno común, puede variar ente 8, 16, 32 pese a que teóricamente debería

poder permitir la conexión de hasta 254 máquinas, número que corresponde a las direcciones IP existentes.

3.3. Software

3.3.1. Sistema operativo Ubuntu 12.04

Para este proyecto se empleará el sistema operativo *Ubuntu* 12.04 basado en *Linux*. Los motivos de la utilización del mismo son múltiples, entre los que destaca el hecho de que el software mediante el cual se va a desarrollar el algoritmo tiene las versiones más estables para dicho sistema operativo como se verá en el siguiente punto. Por otro lado, hay que añadir que cuando se comenzó a realizar este proyecto, la versión 14.04 de *Ubuntu* no era estable y la anterior a esta, la 13.04, no estaba dando los resultados deseados. Es por eso que se va a emplear *Ubuntu* 12.04.

A esto hay que añadir que las computadoras conectadas a los sensores van a tener que soportar el mismo software de desarrollo (ROS) que el ordenador local desde donde se realizará toda la programación, necesario para poder compatibilizar completamente las mismas funcionalidades en todos los dispositivos dentro de las limitaciones aportadas por las diferencias técnicas entre las máquinas.

Ubuntu entra dentro del modelo de desarrollo conocido como Software libre siendo por tanto de código abierto. Esto permite su uso, copia, modificación e incluso redistribución. Como se dijo anteriormente, cuando se hablaba de las *Raspberry Pi*, estas iban a usar una versión de *Raspbian* modificado que incluía el software ROS. Este puede ser un ejemplo de las posibilidades que ofrece el hecho de usar software libre como es *Raspbian*, al igual que lo es *Ubuntu*, ambos procedentes de *Linux*.

3.3.2. *Robot Operating System* (*ROS*)

Introducción y reseña histórica

ROS (*Robot Operating System*) o Sistema Operativo Robótico, es un *framework* empleado para el desarrollo de software para robots que proporciona la funcionalidad de un sistema operativo. El origen del desarrollo de ROS comenzó en 2007 bajo el nombre de *switchyard* por el Laboratorio de Inteligencia Artificial de la Universidad de Stanford con el objetivo de dar soporte al proyecto STAIR (*STanford Artificial Intelligence Robot*) y al RP (*Personal Robot*), consiguiendo la creación de prototipos de sistemas software dinámicos y flexibles orientados para usos robóticos. Será en 2008 cuando el desarrollo de ROS continúe en *Willow Garage*, instituto de investigación robótica donde se congregan más de veinte instituciones, el cual proporcionó

abundantes recursos para extender los conceptos ya creados y poder llegar a implementaciones revisadas y probadas.

Figura 3.4: Zona superior izquierda logo de ROS, zona inferior izquierda logo de *Willow Garage*, a la derecha robot STAIR

ROS proporciona los servicios estándar de un sistema operativo tales como hardware de abstracción, control de dispositivos de bajo nivel, implementación de funcionalidades más comúnmente usadas, paso de mensajes entre procesos y gestión de paquetes. Se encuentra basado bajo una arquitectura de grafos tomando lugar el procesamiento en los nodos, los cuales pueden mandar, recibir y multiplexar mensajes de sensores, control de estados, planificaciones y actuadores, entre otras funciones. ROS y sus librerías están desarrollados para un sistema UNIX (*Linux*) aunque su extensión a otros sistemas operativos está en proceso, existiendo para estos últimos versiones consideradas como experimentales.

Por dar una visión algo más global acerca de ROS, podría considerarse que se divide en dos partes. En primer lugar aquella que compone el sistema operativo descrito anteriormente y, en segundo lugar, *ros-pkg*, conjunto de paquetes disponibles debido a aportaciones de usuarios que implementan diferentes funcionalidades como percepción, localización y mapeo simultáneo, simulación, etc.

Finalmente, cabe mencionar el objetivo principal de ROS: apoyar la reutilización de código fuente en la investigación y desarrollo de la robótica, pudiéndose comparar con un sistema de código repositorio donde se permite la colaboración entre diferentes usuarios.

Características

En este apartado se va a tratar de dar una idea global de las diferentes características técnicas y funcionalidades más importantes que ROS puede aportar [13].

- Comunicación ROS es un software que actúa como *middleware*, es decir, al más bajo nivel proporciona asistencia a aplicaciones para interactuar o comunicarse con otras, y no sólo aplicaciones sino también paquetes de programas, redes, hardware, entre otros. Esto ofrece una simplificación a la hora de programar ya que el hecho de generar conexiones y sincronizarlas en sistemas distribuidos, como es el caso de este proyecto, puede llegar a ser bastante compleja. Entre las facilidades cabe destacar:
  - Comunicación síncrona mediante servicios entre Cliente/Servidor, empleando llamadas a dichos procedimientos remotos.
  - Transmisión de datos asíncrona a través de temas (*topics*) entre Publicador/Subscriptor.
  - Almacenamiento de mensajes y posibilidad de su posterior reproducción.
- Herramientas
  - Rviz: proporciona visualización en tres dimensiones de los datos adquiridos por diferentes sensores además de la posibilidad de visualización de cualquier robot de la forma URDF.
  - Rqt: infraestructura de ROS creada en forma de *plugins* con el objetivo de permitir el desarrollo de interfaces gráficas en forma de *plugins* para robots. Junto a la extensa biblioteca de plugins que ROS proporciona, cada usuario puede personalizar e incluso crear nuevos componentes para rqt. Uno de los *plugins* más útiles, y empleado en el desarrollo de este proyecto como se verá más tarde, es *rqt_graph*. Esta aplicación permite visualizar un sistema ROS creado, en ejecución. De esta manera se puede ver desde una perspectiva más gráfica los diferentes nodos, las conexiones, así como los temas (*top*ics, concepto explicado posteriormente) que se estén empleando. Los beneficios que ofrece son claros, ya que depurar y entender errores de forma visual es mucho más sencillo que en código. Existen además otros *plugins* aunque no se van a emplear en este trabajo, estos son *rqt_plot* y *rqt_topic*, entre otros, que aportan funcionalidades más concretas tales como supervisión de voltajes o cualquier variable dependiente del tiempo y control de temas publicados en el sistema, respectivamente.

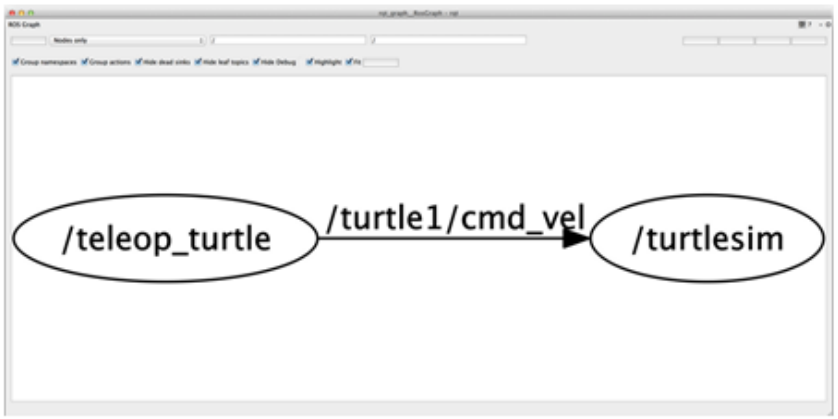

Figura 3.5: Ejemplo de gráfica generada por *rqt_graph*

Como se acaba de mostrar, *rviz* y *rqt* son dos herramientas gráficas que poseen especificaciones muy interesantes para el desarrollo de un proyecto. Sin embargo, ROS dispone de más de cincuenta herramientas activadas desde línea de comandos que permiten trabajar sobre un sistema sin necesidad de interfaz gráfica.

- Características especiales ROS. Como ya se ha dicho, ROS es un *middleware* que va a facilitar en gran medida los establecimientos de comunicación entre nodos, así como su previa creación y mantenimiento. Pero ROS, además de estas opciones, proporciona una serie de características de gran utilidad que pueden incrementar las propiedades del robot en cuestión. Algunas de estas características ofrecidas por ROS son:

  - Mensajes entre robots: ROS proporciona diferentes formatos de mensajes a través de los cuales se cubren la mayor parte de los casos más usados en robótica, estos son mensajes referidos a sensores y datos de navegación, entre otros.

  - Librería de geometría: Uno de los aspectos imprescindibles a la hora de poder darle autonomía a un robot y poder controlar sus movimientos, refiriéndonos a robots con articulaciones, es el hecho de conocer la posición de las mismas en todo instante. ROS incluye una librería capaz de realizar dicha estimación e ir actualizándola.

  - Análisis: ROS ofrece la posibilidad de realizar diagnósticos sobre robots, para localizar problemas y poder solucionarlos.

  - Recopilación de datos: A través del paquete *Rosbag*, ROS proporciona la capacidad de almacenamiento de mensajes a través de los diferentes temas que componen un sistema.

Conceptos básicos

En este apartado se va a tratar de dar una breve definición de diferentes conceptos que sirven para constituir un sistema en ROS. No se darán detalles en profundidad ya que eso se realizará en posteriores capítulos cuando se proceda a realizar la implementación.

- Nodos Son procesos donde se realiza la carga computacional. ROS está diseñado para crear sistemas modulares llegando a dividirse a pequeña escala: un sistema está típicamente compuesto de varios nodos. En este contexto, el término nodo es intercambiable con módulo software. El uso de dicho concepto surge de observaciones de sistemas ROS ejecutándose en tiempo real: cuando muchos nodos se están ejecutando, se pensó conveniente representar la red de comunicaciones entre nodos como un grafo. Para realizar las labores de comunicación entre los nodos de un sistema se emplean los mensajes a través de temas o la llamada a peticiones de servicios remotas.

  Para programar los nodos se emplean las librerías *roscpp* o *rospy*, dando la posibilidad de programar en C++ o en *Phyton*.

- Maestro: RosCore

  Servicio principal y esencial para poder hacer funcionar un sistema bajo ROS. Consiste en registro de nombres y de consulta para el resto de la computación. Almacena temas y servicios de los nodos. Dichos nodos se comunican a través del *Roscore* pudiendo recibir información de otros nodos, establecer conexiones, etc. Así mismo, el Maestro, puede comunicarse con los nodos cuando hay cambios en el registro de información, lo que permite el establecimiento dinámico de conexiones a medida que nuevos nodos se ejecutan.

- Mensajes

  Como se adelantó anteriormente, el paso de mensajes es el medio mediante el cual se comunican los nodos. Los mensajes tienen un tipo concreto de estructura, en ROS vienen definidos de tipo entero, flotante, booleano,etc. pudiendo llegar a contener estructuras anidadas y arrays. A esto se le suma la opción de crear mensajes personalizados, funcionalidad que se emplea en este proyecto como se verá en la implementación.

- Temas: *Topics*

  Los nodos publican mensajes en un *topic* dado, que consiste simplemente en una cadena con el nombre que se desee para identificar el contenido de los mensajes. Un nodo que esté interesado en una serie de datos en especial, se subscribirá al *topic* apropiado. Puede haber varios nodos publicadores y subscriptores en un mismo *topic* y también existe la posibilidad de la existencia de un solo nodo publicando o subscrito a un solo *topic*.

- Servicios

  Aunque el sistema publicador-subscriptor empleando los *topics* es un modelo muy flexible de comunicación, su esquema de transmisión de datos no es apropiado para sistemas síncronos, que pueden simplificar el diseño de algunos nodos. En ROS esto es conocido como servicio, definido con un nombre y un par de mensajes tipo: uno para la petición y otro para la respuesta.

Análisis del software

En la figura 3.6 podemos observar la constitución de la configuración de una red típica en ROS. Claramente, se puede observar una relación directa entre este sistema y el que se va a crear en este proyecto.

Como se dijo con anterioridad, Ubuntu va a ser el sistema operativo bajo el que se va a trabajar, concretamente la versión 12.04. Así mismo las *Raspberry Pi* tendrán instalado *Raspbian*. Uniendo ambas distribuciones de *Linux* la opción que se consideró más viable en cuanto a ROS se refiere fue ROS *Fuerte*.

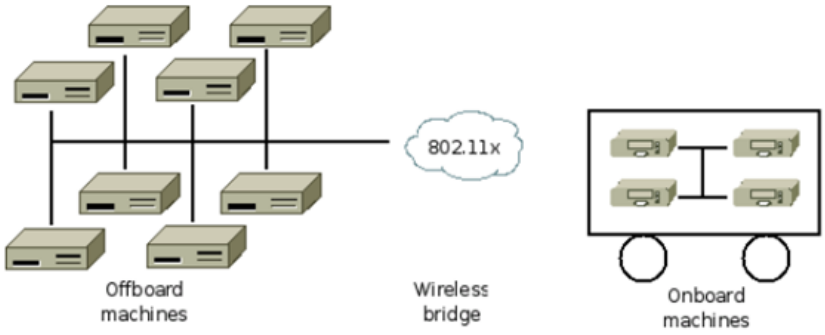

Figura 3.6: Configuración de una red típica en ROS

3.4. Arquitectura

Llegados a este punto se puede realizar un pequeño esquema en el que se pueda tener una visión genérica de la arquitectura del sistema de la cual se va a partir. Como podemos observar en la figura 3.7, la arquitectura constará de un puesto de adquisición de datos el cual se comunicará por SSH con cada uno de los nodos mediante la red local creada por el *router WiFi*.

Inicialmente se puede realizar una clasificación entre nodos completos (nodo móvil y nodos fijos) y nodos incompletos (nodos fijos incompletos). En el caso de estar los nodos fijos completos, significa que cada uno de ellos podrá medir distancias respecto al resto de nodos que se encuen-

tran en la misma red de sensores e ir transmitiendo dicha información al nodo móvil, también completo que, además de realizar las mismas funciones de medida que el resto, se encargará de recoger todos los datos tomados para posteriormente transmitirlos al puesto de trabajo. Por otro lado, se disponen de nodos fijos incompletos que van a tener un caracter pasivo en el sistema ya que, pese a poder medir distancias con el resto de nodos, no está dotado de capacidad de transmisión de datos por lo que su principal función será de baliza pasiva.

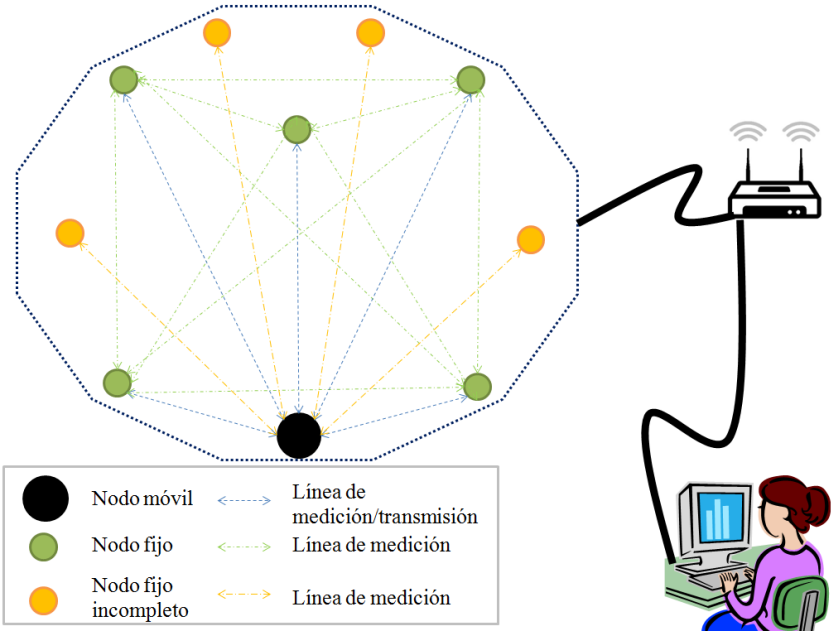

Figura 3.7: Arquitectura del sistema

En capítulos posteriores se explicará detalladamente las diferentes configuraciones a realizar para conseguir un correcto funcionamiento de la arquitectura aquí planteada.

Capítulo 4

Descripción del método implementado

4.1. Introducción

Este capítulo va a tratar de explicar fundamentalmente el algoritmo implementado que da una posible solución al problema de comunicación y adquisición de medidas de distancia entre los diferentes nodos que componen la red sensorial del sistema. Especialmente se van a exponer los diferentes caminos que se han ido tomando, comentando los motivos de las decisiones tomadas.

Como se dijo en capítulos anteriores, el sistema va a estar formado por N nodos compuestos por un sensor *Nanotron NanoPAN 5375* y una *Raspberry Pi*, siendo N el número de balizas empleadas. Inicialmente se buscaron soluciones donde solamente existían dos nodos para simplificar el problema, realizando finalmente un algoritmo genérico dependiente de N nodos y otros factores.

Realmente, el problema principal consiste en establecer comunicación entre todas las *Raspberry* que componen el sistema ya que para crear la red sensorial simplemente habría que realizar algunas modificaciones en los diferentes códigos, así como el acoplamiento de los sensores a estas pequeñas computadoras. Separando inicialmente el sensor de la *Raspberry* que compone cada nodo, se va permitir trabajar para completar el primer objetivo que consiste en establecer comunicación bidireccional entre cada uno de los dispositivos.

Como la mayor parte los problemas que se presentan en tecnología y en prácticamente todos los aspectos de la vida, existen numerosas soluciones viables. Este proyecto ha sido enfocado desde un punto de vista que intenta dar una solución genérica a la par que adaptada al sistema del que se parte, con el objetivo de obtener un resultado óptimo en cuanto a eficiencia se refiere.

Resumiendo, una vez explicado en el capítulo anterior cada una de las plataformas que van a componer el sistema, en este apartado se va a intentar comentar cada una de las soluciones tomadas tanto en el ámbito programación como en cuanto a hardware se refiere.

4.2. Descripción de la solución adoptada

4.2.1. Características

Una vez planteada la arquitectura del sistema y cada uno de los componentes que la forman, hay que tratar de pensar de manera general una serie de premisas que van a tener gran importancia para lograr cumplir el objetivo de creación del algoritmo de comunicación entre los diferentes nodos que componen la red sensorial. Estas características son:

- Escalabilidad: se pretende obtener la habilidad de reacción y adaptación del sistema, así como la capacidad para hacerse más grande sin perder calidad. Reflejado en el sistema en cuestión, se desea poder trabajar con un número N de nodos sin perder las propiedades que caracterizan al sistema, independientemente de si N es un número elevado o no.
- Modularidad: se trata de dar la posibilidad al algoritmo de ser dividido en partes más pequeñas para trabajar independientemente con cada una de ellas pero que tienen conexiones con otros módulos.
- Comunicación Asíncrona: se trata de un sistema donde la transmisión de datos no va a estar determinada por ningún tipo de señal o ciclo de reloj. Es un sistema asíncrono donde el envío y recepción de datos va a estar marcado por un cierto orden, realmente aleatorio, que se deberá implementar en el algoritmo .
- Tiempo de transmisión: Al estar trabajando en red y además al ser los sensores de medida de tipo ToF, van a existir una serie de pérdidas que habrá que tener en cuenta para poder reducirlas

Finalmente cabe destacar que este trabajo ha sido desarrollado empleando un repositorio Git en la nube para control de versiones. En particular se ha utilizado *Bitbucket*, en la figura 4.1 se puede observar una captura de este servicio.

El proyecto se denomina ***interbeacon*** y parte de una serie de implementaciones, concretamente de aquellas que habilitan la conexión de los sensores permitiendo la toma de medidas como se verá posteriormente.

4.2.2. Envío y recepción de mensajes

El envío y recepción de mensajes entre los nodos fijos y el móvil es una de las tareas más importantes, por no decir la mayor relevancia que va a intervenir en el desarrollo de este trabajo. Una vez establecidas las conexiones, el siguiente paso es dotar a los nodos de dicha funcionalidad de transmisión de datos. Sin embargo, se ha considerado que la descripción del envío y recepción de datos iba a ser más importante tenerla clara antes de proceder a la creación de la red.

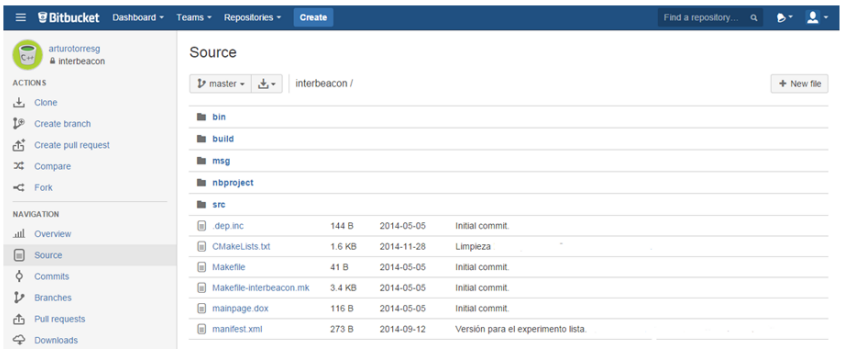

Figura 4.1: Proyecto *interbeacon* en Bitbucket.

Como ya se comentó en el capítulo 3, ROS iba a ser el *framework* para el desarrollo de este proyecto. También se explicaron algunos de los conceptos básicos del entorno de ROS que a continuación se van a tratar con mayor detenimiento.

Una de las primeras decisiones a tomar será respecto al mecanismo de intercambio de datos. Partiendo de la premisa de que el sistema no va a seguir ninguna estructura temporal (sistema asíncrono), parece claro llegar a la conclusión de que el método más correcto para realizar el envío de datos es a través de mensajes entre nodos, empleando para ello *topics*.

Por otro lado, se podría pensar en el empleo de servicios entre Cliente/Servidor, realizando llamadas a procedimientos remotos. En ese caso, un cliente realizaría una petición de servicio y esperaría la respuesta del servidor síncronamente. Pese a ser el sistema asíncrono, con este método se podría también alcanzar los objetivos deseados debido a la capacidad de comunicación bidireccional que proporciona.

Para poder comprender correctamente el funcionamiento de ambas formas, se realizaron diferentes pruebas con ROS *Fuerte* creando dos nodos que se comunicaban localmente en el mismo PC.

Para el primer caso comentado, uno de los nodos actuaba de cliente y el otro de servidor. Extrapolando de dos nodos a N, siendo uno de ellos el nodo móvil, se tanteó la posibilidad de que existieran N-1 servidores (nodos fijos) y un cliente que correspondiera al nodo móvil. Conforme fuera avanzando el algoritmo, el cliente iría solicitando el servicio de toma de medidas y envío de datos a cada uno de los servidores, dándole posteriormente respuesta al cliente.

Pese a ser una solución completamente viable, finalmente, la opción Cliente/Servidor quedó descartada debido a la poca flexibilidad que aportaba al sistema al tener que ir realizando peticiones de servicio servidor a servidor, ya que es síncrono, se añadiría una deriva en las sucesivas tomas de medidas que distorsionaría la medida real.

Sin embargo, la realidad es que, teóricamente los servicios son idóneos para sistemas distribuidos donde las interacciones entre elementos vienen determinadas por una petición con su

respuesta correspondiente, como es el caso en cuestión.

A continuación se adjunta un pequeño ejemplo que se probó, extraído de uno de los tutoriales que proporciona este software libre en su propia *wiki* creada por diferentes aportaciones de desarrolladores.

Ejemplo Cliente/Servidor:

http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28c%2B%2B%29

A través de este problema sencillo se pueden observar de una forma más práctica las características de este formato de comunicación.

Como se puede observar, pese a estar ante un ejemplo simple (petición de suma de dos números, respuesta con el resultado), la flexibilidad que esto puede aportar está realmente limitada en el sentido de que tener N servidores síncronamente trabajando no garantiza la eficiencia temporal necesaria. Hay que tener en cuenta que el robot móvil va moviéndose entre los nodos tomando medidas continuamente. Si el robot tiene que ir solicitando servicios nodo a nodo y esperando su respuesta, la pérdida de flexibilidad se hace más presente aún, añadiendo tiempo en la obtención de medidas, las cuales diferirán eventualmente respecto a la posición actualizada del robot.

Dejando parcialmente descartado este método, se comienza a estudiar el envío de datos utilizando mensajes a través de *topics*, publicadores y subscriptores.

Al igual que en el caso anterior, se partió de dos nodos locales pero ahora uno de ellos actuaría de publicador y el otro de subscriptor. La idea es algo más sencilla que el caso anterior ya que, como se comentó, este tipo de paradigma de comunicación no está diseñado para comunicación bidireccional entre nodos.

Sin embargo, la versatilidad aportada por este sistema es tan grande que la creación de comunicación bidireccional es posible como se verá explícitamente en el capítulo siguiente. Por tanto, en una primera aproximación y tras diferentes pruebas, este método va a permitir solventar el problema de petición-respuesta que Cliente/Servidor tenía resuelto. Además, al ser un conjunto asíncrono, no va a existir una deriva tan elevada en la transmisión de datos ya que en todo momento, extendiendo a N nodos, cada uno de ellos estarán típicamente comunicados entre sí y mandando datos al nodo móvil sin seguir un orden establecido. Por tanto, si alguno de ellos no está disponible el resto seguirá enviando y, cuando éste recobre la capacidad de transmisión, volverá a enviar datos.

El ejemplo del cual se parte para obtener una idea global de este método se extrajo así mismo de uno de los tutoriales que proporciona ROS en su web

Ejemplo Publicador/Subscriptor:

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29

Ahora, el publicador envía un mensaje que contiene una cadena de caracteres a través de un *topic* al cual está subscrito el subscriptor. Cuando el subscriptor recibe el mensaje, realiza la llamada a una rutina la cual trata dicho mensaje recibido. En este caso, al recibir la cadena, la función realiza la impresión por pantalla.

En cada *topic* solamente se publican mensajes de tipo predefinido, en el ejemplo son de tipo cadena de caracteres. ROS proporciona una gran variedad de ellos y a su vez permite la creación de mensajes personalizados que realmente serán los usados en la implementación para satisfacer las propiedades del sistema.

Observando las características encontradas, el método que se ha decidido seguir para cumplir las labores de comunicación, en cuanto a envío de datos se refiere, es el de *topics*, publicador y subscriptor.

4.2.3. Establecimiento de conexión

Una vez elegida la forma a través la cual los nodos se van a enviar mensajes, parece un momento oportuno para describir cómo se van a establecer las conexiones. Como se dijo en el capítulo anterior, ROS es una plataforma de software que facilita en gran medida la creación de redes locales.

Existen numerosas formas para enviar datos en una red, y cada una de estas formas tiene sus ventajas e inconveniente, en función de la aplicación a la que vaya destinada. TCP uno de los protocolos más empleados ya que proporciona un flujo de comunicación simple y fiable. Los paquetes TCP siempre llegan con un cierto orden, siendo los paquetes perdidos reenviados hasta que llegan al destino. Pese a ser un protocolo muy eficiente para redes conectadas por cable, todos estas características se vuelven errores cuando la red subyacente es una conexión WiFi con pérdidas. En esta situación, el protocolo UDP es más adecuado. Cuando varios subscriptores se agrupan en una solas subred, puede ser más eficiente para el publicador comunicares con todos ellos de forma simultánea empleando UDP *broadcast*.

Por ello, ROS no se compromete únicamente a un método de comunicación. Cada transporte tiene su propio protocolo para intercambiar datos empleando o TCP o UDP.

ROS posee una serie de condiciones para poder configurar una red. En primer lugar debe existir conectividad bidireccional entre todas las máquinas en todos los puertos que estén conectadas. Y en segundo lugar, cada máquina debe tener preconfigurada un nombre y una dirección IP que permita identificarla del resto.

En primer lugar hay que realizar la configuración de los adaptadores WiFi que van a permitir a los nodos formar parte de la red inalámbrica. Para realizar todas las modificaciones en cada *Raspberry Pi*, se emplea el protocolo SSH (*Secure Shell*) pudiendo realizar los cambios sin la necesidad del empleo de un monitor secundario. Una vez configurados dichos adaptadores, hay

que darle nombre y asociarle una dirección IP a cada máquina para cumplir los requerimientos que ROS impone. Hecho esto y alguna configuración más que se verá más adelante referente a la propia configuración de ROS, se permitirá llegar a la arquitectura comentada en el capítulo anterior.

4.2.4. Descripción general

Aunque en el siguiente capítulo se vaya a realizar una explicación detallada de la implementación, a continuación se va a tratar de comentar qué es lo que realiza el algoritmo y cómo lo hace de forma descriptiva.

Todo comienza con la activación del *router WiFi* que constituirá la red. Cada uno de los sensores se conecta a su respectiva *Raspberry* que previamente tiene realizada toda la configuración tanto de conexión en red mediante un adaptador WiFi, como del algoritmo implementado. En este momento se tienen todos los nodos activos y conectados a la misma red, pero aún sin comunicación entre ellos.

A continuación, desde el puesto de control, el cual también forma parte de la red, se ejecutan diferentes códigos que permitirán activar el sistema completo. Desde estos códigos se van ejecutando los diferentes programas enviando comandos vía SSH a cada una de las *Raspberry Pi*. Tras diferentes envíos se consigue crear la red sensorial en ROS, permitiendo que todos los nodos puedan comunicarse entre sí a través de la misma red WiFi y, principalmente, pudiendo enviar cada uno de los nodos fijos las medidas que van tomando al nodo móvil. Finalmente, se procede a ejecutar el nodo '*static*' en los nodos fijos y el nodo '*mobile*' en el nodo móvil.

Se parte de un archivo el cual presta información acerca de qué nodos están operativos, si los nodos están completos (sensor + *Raspberry Pi*) y, si hay una posición inicial conocida de los mismos, las coordenadas de las mismas.

El nodo móvil realiza lo siguiente:

- Lee el archivo, inicializa sensor y obtiene la identificación que caracterizará al nodo móvil. A continuación adquiere propiedades de Publicador en el *topic* '*Upstream*' y así mismo realiza funciones de Subscriptor creando un vector de Subscriptores al *topic* '*Downstream*'.

- Realizando las tareas de Publicador, el nodo móvil crea un archivo con la medida de distancia entre él mismo y el resto de nodos fijos. Así mismo, si de la lectura del archivo inicial se obtiene que un nodo está completo, el nodo móvil publica en el *topic* '*Upstream*' la identificación del nodo fijo en cuestión.

- Cuando actúa de Subscriptor, se subscribe al tema correspondiente tantas veces como nodos fijos haya, y es por eso que se crea como un vector de dimensión variable. Su labor es la siguiente: abre el archivo creado con las medidas y lo modifica añadiendo las nuevas

medidas publicadas en el *topic* '*Downstream*' al cual está subscrito. La implementación rigurosa se verá próximamente.

Cada uno de los nodos fijos realiza lo siguiente:

- Lee el archivo, inicializa el sensor y obtiene la identificación del nodo fijo. En función de dicha identificación va creando un vector de Publicadores que enviarán los mensajes de medidas a través del *topic* '*Downstream*'. A su vez, hace de Subscriptor del *topic* '*Upstream*' a través del cual el nodo móvil va mandando la identificación de los nodos activos.
- Al recibir un mensaje con dicha identificación, se comprueba qué nodo es y se procede a realizar toda la toma de medidas entre el nodo recibido y el resto, incluido el nodo móvil. Esto lo irán haciendo cada uno de los nodos paralelamente.
- Cuando ya obtienen las medidas, mandan un mensaje a través del *topic* '*Downstream*' que recibirá uno de los Subscriptores localizado en el nodo móvil como se dijo anteriormente, el cuál añadirá dichas medidas en el archivo creado como también se comentó en el apartado previo.

4.3. Conclusión

En este momento del desarrollo del proyecto se ha conseguido explicar todo como conjunto, enlazando los elementos de hardware con los de software así como los procedimientos que va a seguir el algoritmo en su ejecución. Se ha podido comprobar que las posibilidades que ROS proporciona son muy interesantes, destacando como característica principal la adaptabilidad que posee según el tipo de aplicación que se esté desarrollando.

En el caso de este trabajo, como se ha comentado en el presente capítulo, se han tomado decisiones en cuanto al envío de datos y establecimiento de conexión bajo una serie de características o premisas que condicionan al sistema. Todas estas conclusiones han sido desarrolladas de manera descriptiva sin entrar en detalles referentes a configuración y características intrínsecas de los diferentes código implementados. Para ello se dedica el capítulo 5.

Capítulo 5

Detalles de implementación

5.1. Introducción

A lo largo de este capítulo se va a desarrollar de manera extendida cada uno de los aspectos que se han ido tratando a lo largo de esta memoria. Concretamente se va a realizar una explicación detallada de los tres campos claves que van a permitir la ejecución del sistema correctamente.

La estrategia seguida para realizar la implementación se comentó en anteriores capítulos y fue la siguiente: inicialmente se partió de dos nodos simples creados localmente en una misma computadora. Tras lograr la comunicación entre ellos, se dispuso de dos *Raspberry Pi* las cuales tuvieron que ser configuradas como se explicará en este capítulo. Realizadas todas las configuraciones, se estableció la red inalámbrica con la creación de una red local a partir de un *router WiFi*. Conseguido este objetivo, el siguiente paso se componía del desarrollo del algoritmo genérico propiamente dicho, extrapolando a N nodos. En este momento, la siguiente meta consistía en ir comprobando el correcto funcionamiento del sistema completo; de dos nodos se pasó a cinco, posteriormente a diez y finalmente a veinte.

Para lograr alcanzar todo esto se tuvieron que seguir los diferentes pasos de forma ordenada, con el fin de poder depurar errores de la forma más eficaz posible. En primer lugar, la preparación del entorno de trabajo, aspecto primordial para poder ejecutar los diferentes programas de forma remota, así como para conseguir el establecimiento de las conexiones necesarias que permitirá posteriormente crear la red de sensores inalámbrica. Finalmente, se va a proceder a detallar la implementación en ROS, tratando temas tanto de bajo como de alto nivel, entre los que destacan la creación de mensajes personalizados, la configuración de la red, así como la herramienta gráfica de ROS, *rqt*, ya conocida.

5.2. Preparación del entorno de trabajo

Para poder comenzar a realizar configuraciones de Hardware es necesario que el ordenador local donde se va a trabajar, el puesto de operaciones por así decirlo, debe tener instalado *Ubuntu* 12.04 y ROS *Fuerte*. Realizadas estas configuraciones previas, se procede a seguir con el resto de aspectos que integran el sistema.

5.2.1. Instalación y configuración de *Raspberry Pi*

Todas las modificaciones en cada uno de los dispositivos se realizarán vía SSH, en un principio empleando conexión por cable Ethernet. Una vez configurados los adaptadores WiFi, dichas modificaciones se realizarán así mismo por SSH pero empleando la conexión inalámbrica recién establecida.

Raspbian

Las *Raspberry Pi* son pequeñas computadoras que no poseen una memoria física integrada que permita el almacenamiento de datos o donde se pueda instalar el sistema operativo. Es por ello que se disponen de tarjetas SD que se pueden acoplar. En dichas tarjetas de memoria, empleando un programa, se va a incorporar la imagen del sistema operativo. Como se comentó anteriormente, la variedad de plataformas que pueden ser instaladas en las *Raspberry Pi* es relativamente amplia pero para este trabajo se decidió por una modificación personalizada de *Raspbian* llamada *ROSbian*. Dicha personalización consiste en la incorporación, en la propia imagen del sistema operativo, del Software principal de este proyecto, ROS *Fuerte*.

El programa mencionado es *Win32 Disk Imager*. Su función principal es escribir una imagen de disco en un dispositivo de almacenamiento extraíble o viceversa, copia de seguridad a un archivo de imagen. Para el caso de este proyecto, será necesario para escribir la imagen de *ROSbian* en la tarjeta SD. En la figura 5.1 a continuación se observa que la imagen es '*ROSbian16.img*' y será copiada pulsando el botón '*Write*' al dispositivo que indica el campo '*Device*'.

Identificación de cada *Raspberry Pi*

Al tener cada una de las *Raspberry Pi* un identificador de red común conectadas a una misma red local con la misma distribución de *Linux*, pueden suceder diferentes problemas de incompatibilidad, por ello es necesario modificar dicho nombre realizando los pasos que se muestran a continuación. Dicho identificador de red se conoce como *hostname*, nombre con el que aparecerá en el listado de dispositivos conectados a la red local. Si no se realiza dicha modificación, sólo una de las *Raspberry Pi* podría conectarse debido a que aparecería un conflicto con los nombres de host. Como se dijo anteriormente, el primer paso siempre será realizar la conexión mediante

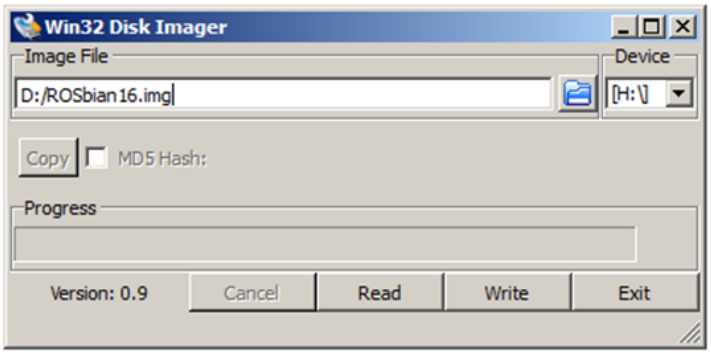

Figura 5.1: Captura de pantalla del software *Win32 Disk Imager.*

SSH.

Los pasos a seguir son los siguientes:

– Se introduce el siguiente comando para editar el archivo 'host':

```
$ sudo nano /etc/host
```

En la pantalla emergente se modifica únicamente la siguiente línea: 127.0.1.1 raspberrypi (cambiando el nombre por el nuevo deseado)

– Ahora se edita el archivo 'hostname' mediante la siguiente línea:

```
$ sudo nano /etc/hostname
```

En la pantalla emergente se modifica el nombre que aparece por el nuevo deseado (debe coincidir con el anterior)

– Por último, para hacer efectivos los cambios se deben ejecutar las siguientes líneas:

```
$ sudo /etc/init.d/hostname.sh
$ sudo reboot
```

En este momento cada una de los dispositivos tendrá su nombre, faltando únicamente como configuración la asociación de una dirección IP para la conexión inalámbrica que se verá en el siguiente punto.

Configuración de adaptadores WiFi

Para realizar la configuración de cada uno de los adaptadores hay que seguir una serie de pasos que se comentarán a continuación. Como requisitos previos se necesitan un cable Ethernet y el adaptador WiFi conectado a uno de los puertos USB de los que dispone la *Raspberry Pi.*

Se sabe que cada dispositivo conectado a la red tiene asociada una dirección física MAC y una dirección IP, la primera es única para cada dispositivo Hardware y la segunda es típicamente variable.

Al activar el *router WiFi* y conectar el PC local al mismo introduciendo la contraseña previamente establecida, se puede acceder a la ventana de configuración del router introduciendo la dirección 192.168.1.1 en la barra de direcciones del navegador. Cada vez que se conecte un dispositivo se podrá observar su MAC y la IP dinámica generada.

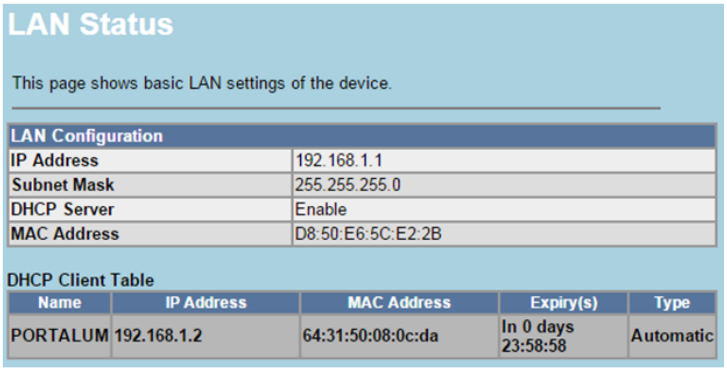

Figura 5.2: Captura de pantalla de la configuración del router.

Se introduce el siguiente comando para visualizar todos las interfaces activas:

```
$ ifconfig
```

Los pasos a seguir a partir de aquí son los siguientes:

– Se crea el archivo *wpa_supplicant.conf* donde se añadirá toda la información referente al router.

  Se crea el archivo:

  ```
  $ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
  ```

  Se añaden las siguientes líneas:

  ```
  ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
  update_config=1
  network={
          ssid="ASUS"
          proto=RSN
          key_mgmt=WPAPSK
          pairwise=CCMP TKIP
  ```

```
        group=CCMP TKIP
        psk="PASSWORD"
}
```

– El último paso es añadir la referencia a la interfaz conectada conocida gracias al comando *ifconfig* antes expuesto.

  Se abre el archivo *interfaces* con el siguiente comando:

```
$ sudo nano /etc/network/interfaces
```

  Se añaden las siguientes líneas que asocian el adaptador WiFi con la conexión inalámbrica aportada por el *router WiFi*.

```
allow-hotplug wlan0
wlan0
iface wlan0 inet dhcp
wpaconf/etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

– Sólo queda reiniciar la red ejecutando el siguiente comando, así mismo se considera oportuno reiniciar la *Raspberry Pi* desconectando en ese momento el cable Ethernet.

```
$ sudo /etc/init.d/networking restart
```

En este momento queda configurado el adaptador, conectándose automáticamente al router predefinido. Una vez se ha dotado a cada dispositivo de esta propiedad y de la identificación dada por el *hostname*, es necesario ubicar cada dispositivo en la red. De esta forma cuando posteriormente se deseen enviar paquetes de datos no habrá que ir a la configuración del WiFi para obtener la nueva dirección IP dinámica generada.

Es por eso que, empleando la MAC de cada adaptador WiFi se puede crear una dirección IP estática asociada, *DHCP static*, lo cual es esencial para el correcto funcionamiento del sistema. Se llega a obtener lo que se observa en la figura 5.3.

Correspondiendo cada uno de los dispositivos configurados a una *Raspberry Pi* que hará de nodo fijo junto con su sensor respectivo enumerado de la misma forma. Uno de estos nodos, concretamente el 198.168.1.100 será el nodo móvil. En esta imagen también se pueden observar los nuevos *hostnames* creados en el punto anterior. Ahora queda todo listo para comenzar el desarrollo de la implementación propiamente dicha, pero antes se realizará una última configuración referente a la hora y fecha de cada dispositivo.

DHCP Client Table:

Name	IP Address	MAC Address
raspberrypi18	192.168.1.118	80:1F:02:D6:12:8A
raspberrypi17	192.168.1.117	80:1F:02:AB:AA:E5
raspberrypi16	192.168.1.116	80:1F:02:AB:9D:25
raspberrypi15	192.168.1.115	80:1F:02:D6:15:16
raspberrypi14	192.168.1.114	80:1F:02:AB:9D:1D
raspberrypi11	192.168.1.111	80:1F:02:D6:12:8D
raspberrypi13	192.168.1.113	80:1F:02:D6:12:80
raspberrypi12	192.168.1.112	80:1F:02:D6:12:5C
raspberrypi10	192.168.1.110	80:1F:02:D6:12:8E
raspberrypi9	192.168.1.109	80:1F:02:D6:12:66
raspberrypi8	192.168.1.108	80:1F:02:D6:15:14
raspberrypi7	192.168.1.107	80:1F:02:D6:12:46
raspberrypi6	192.168.1.106	80:1F:02:D6:14:FE
raspberrypi5	192.168.1.105	80:1F:02:D6:15:3F
raspberrypi4	192.168.1.104	80:1F:02:D6:12:7C
raspberrypi3	192.168.1.103	80:1F:02:AB:9D:16
raspberrypi0	192.168.1.100	00:0F:55:A8:A8:6E
raspberrypi2	192.168.1.102	80:1F:02:D6:15:46
raspberrypi1	192.168.1.101	80:1F:02:AB:AA:E2

Figura 5.3: Captura de pantalla de la configuración obtenida *DHCP static*

NTP

Un aspecto sencillo pero realmente necesario es dotar al sistema de sincronización, es decir, que todos los componentes trabajen bajo la misma fecha y hora. Como todo dispositivo electrónico, las *Raspberry Pi* disponen de reloj interno pero no está configurado para conectarse a una red que pueda servirle de servidor NTP. Al trabajar con una red local, el dispositivo *router WiFi* tampoco puede proporcionarle dicha información. Para lograr tener la fecha y hora reales es necesario cogerla de la red.

El ordenador empleado en el puesto de trabajo sí tiene reloj de tiempo real sincronizado con un servidor NTP, por lo que a partir de este el resto de dispositivos serán configurados para sincronizarse con esta máquina.

Los pasos seguidos son los siguientes:

– Instalar paquete NTP en todas las computadoras:

```
$ sudo apt-get install ntp ntpdate
```

– Configurar el servidor NTP en el PC local con IP 192.168.1.2 conectado a la red, modificando el archivo */etc/ntp.conf*

Si las siguientes líneas no existen y no se pueden modificar, se deben crear:

```
server 127.127.1.0
```

```
fudge  127.127.1.0 stratum 10

restrict 192.168.1.2 mask 255.255.255.0 nomodify
```

– Iniciar el servidor ejecutando el siguiente comando:

```
$ sudo /etc/init.d/ntp start
```

– Configurar los clientes NTP, las *Raspberry Pi*, modificando el mismo archivo */etc/ntp.conf* pero ahora cambiando los servidores por el que se acaba de crear:

```
server 192.168.1.2 iburst
```

– Finalmente se inicializa la configuración recién hecha ejecutando:

```
$ sudo /etc/init.d/ntp start
```

5.3. Implementación en ROS

5.3.1. Características técnicas

Antes de comenzar a desarrollar los pasos seguidos en la implementación se van a comentar una serie de aspectos referentes a características técnicas del sistema.

En primer lugar, hay que comentar el contenido de los mensajes que van a mandar tanto el nodo '*mobile*' como los '*static*'. El primero de ellos tiene dos campos, uno referente a la identificación del nodo que se quiere medir y otro que proporciona el máximo rango. Por otro lado, los nodos '*static*', enviarán mensajes con información del nodo origen (respecto al que se toman las medidas), los nodos destino (nodos cuya distancia relativa al nodo origen ha sido medida) y por último las medidas.

En segundo lugar, hay que tener en cuenta el tiempo de ejecución del sistema, el cual dependerá tanto de el coste temporal necesario para la toma de cada medida, como del número de nodos que formen la red. Como se verá en la implementación de los nodos, este parámetro de tiempo será imprescindible para poder concatenar los mensajes sin obtener pérdidas asociadas a mensajes incompletos.

$$T = N \times 15(ms) + X(ms) \begin{cases} \text{N: Número de nodos de la red} \\ \text{15: Coste temporal por medida realizada} \\ \text{X: Holgura, proporciona un cierto margen temporal} \\ \text{T: Coste temporal por ejecución del algoritmo} \end{cases} \tag{5.1}$$

Esta información es necesaria conocerla a priori para poder adecuar las diferentes implementaciones de los nodos y mensajes personalizados.

5.3.2. Creación del *workspace*

Para poder trabajar en ROS es necesario y realmente útil la utilización de un espacio de trabajo o *workspace*. Dentro de este se podrán crear los diferentes paquetes, nodos, mensajes y demás configuraciones. Al emplear ROS Fuerte, el *workspace* es de tipo *rosbuild*, y basta con ejecutar la siguiente línea para crearlo:

```
$ rosws init ~/fuerte_workspace /opt/ros/fuerte
```

Hecho esto se puede proceder a la creación de un directorio que tendrá el nombre *project_1* dentro del cual se creará el paquete *interbeacon* el cual contendrá todos los archivos que constituyen el proyecto.

Se crea el directorio:

```
$ mkdir ~/fuerte_workspace/project_1
```

Cada vez que se cree un nuevo directorio, este debe ser añadido a la variable de entorno llamada $ROS_PACKAGE_PATH, la cual incluye todos los directorios que forman parte del espacio de trabajo. Si no se realiza esta modificación, las herramientas de ROS no conseguirán ubicar los diferentes paquetes que componen los directorios en cuestión.

Es por ello que en el archivo *.bashrc*, el cual compone la configuración del terminal o *bash*, se debe añadir la siguiente línea para poder interactuar por medio de texto en una interfaz no gráfica:

Se abre el archivo:

```
$ sudo gedit ~/.bashrc
```

Se modifica añadiendo:

```
export ROS_PACKAGE_PATH = $ROS_PACKAGE_PATH: /home/carcormir/
fuerte_workspace/project_1
```

Hecho esto, basta con actualizar el archivo obteniendo la inicialización de las variables de entorno, ejecutando:

```
$ source .bashrc
```

En este momento se puede proceder a la creación del paquete *interbeacon*. Cabe mencionar que de no haber realizado las modificaciones en el archivo *.bashrc*, los siguientes pasos no podrían ejecutarse correctamente.

Se cambia al directorio:

```
$ roscd project_1
```

Se crea el nuevo paquete con dos dependencias: la primera de ellas permite la utilización de mensajes de diferentes tipos básicos, y la segunda va a establecer el tipo de programación a emplear, incluyendo las librerías que van a permitir la utilización del lenguaje de programación deseado (en este caso C++) con ROS.

```
$ roscreate-pkg interbeacon std_msgs roscpp
```

En este momento, ya está hecha toda la configuración para poder comenzar a crear los nodos y mensajes.

5.3.3. Mensajes personalizados

Esta herramienta es bastante útil ya que va a permitir tener una mayor libertad dando la posibilidad aprovecharnos de una herramienta de ROS para personalizar y adecuar mejor el tipo de mensajes que se van a enviar. Como se vino diciendo anteriormente se van a tener dos tipos de mensajes. Estos son *activar* y *measure*.

El primero de ellos, *activar*, se compone de una variable entera llamada *id* y una flotante llamada *max_range*. Este tipo de mensajes se enviará a través del *topic* '*Upstream*' por el nodo *mobile* y será recibido por el nodo *static* correspondiente.

Por otro lado se tiene el mensaje *measure*, el cual va a incluir un vector de enteros de nodos origen, *id_source*, un vector de enteros nodos destino, *id_destiny*, y por último un vector de flotantes con las medidas entre dichos nodos, nombrado *measurement*. Estos mensajes serán transmitidos a través del *topic* '*Downstream*' por el nodo *static* hasta el nodo subscrito a dicho *topic* que en este caso será el denominado *mobile*.

Para crear este tipo de mensajes es necesario crear los archivos *.msg* correspondientes

activar.msg	*measure.msg*
int32 id	int32[ ] id_source
float32 max_range	int32[ ] id_destiny
float32[ ] measurement	

Ahora basta con ubicarse en el paquete que se está trabajando y crear un nuevo directorio llamado *msg*. A continuación, se copian los nuevos archivos creados en dicho directorio y se realizan las modificaciones pertinentes en el archivo *CMakefile.txt* como se verá posteriormente. Estas modificaciones serán necesarias para que ROS pueda compilar los nuevos mensajes permitiendo así su utilización dentro de los nodos que se deseen.

Finalmente, para poder utilizar el nuevo tipo de mensajes creados deben incluirse a modo de cabeceras las líneas adjuntas en los nodos que se deseen, como si se trataran de librerías de C++. Hecho esto se pueden crear variables de tipo *activar*, por ejemplo, obteniendo con ello como una especie de estructura compuesta por tres campos.

```
#include "interbeacon/activar.h"
#include "interbeacon/measure.h"
```

5.3.4. Nodos *Mobile* y *Static*

Sin lugar a duda, este punto es uno de los más importantes de todo el proyecto ya que compone la aplicación propiamente dicha. La correcta programación de los nodos va a permitir exprimir al máximo las capacidades del sistema como conjunto, hecho necesario para poder optimizar tanto la calidad como la cantidad de los resultados que se pretenden adquirir.

La idea general sobre las funciones que los nodos iban a tener quedó clara en el capítulo anterior. Se comentó que iba a existir un nodo móvil llamado *mobile*, encargado de medir la distancia entre él mismo y el resto de nodos. Posteriormente avisaría a aquellos nodos fijos, conocidos como *static*, que cumplieran un requisito: el nodo formado por sensor-*Raspberry Pi* esté completo. Inmediatamente después, el nodo fijo realizaría sus tareas de medición, devolviendo dichas medidas al nodo móvil, el cual finalmente crearía un fichero para almacenar todos los datos.

Una vez en contexto, se ha visto oportuno para la comprensión de las labores entre los nodos, la explicación de los pseudocódigos asociados a los nodos *mobile* y *static*. Hay que mencionar que todos los códigos creados para el proyecto deben estar ubicados en el directorio *src*, dentro del paquete que compone la aplicación llamado *interbeacon*.

Pseudocódigo *mobile*

Declaración de librerías:

ROS:Incluye todos los elementos para programar en C++ las diferentes herramientas de ROS.
ROS time: Específica de tiempo, necesaria para poder obtener tiempo de ejecución y hora del sistema, a través de ROS.
Mensajes activar/measure:Mensajes personalizados, necesarias para poder utilizar los mensajes.
Cntronmod: Creación de conexión puerto serie con sensores, inicialización de los mismos, identificación de sensores, realización de medidas.
Generales: Estándar, vectores, cadenas, ficheros.

Definición de variables globales:

outputme, logfile_name:Variables asociadas al archivo creado con las medidas realizadas.
map, pi, x, y, z: Variables para almacenar la información contenida en map.txt
begin: Variable temporal de ROS.

Inicio función chatterCallback: Esta función se activa cuando se publica en el topic 'Downstream' un mensaje de tipo measure procedente de uno de los nodos static.
Apertura de outputme, si no existe lo crea, de lo contrario modifica el existente
Para cada medida tomada
Escribir el tiempo, nodo origen, nodo destino, medida en outputme
Fin para
Cierre de outputme
Fin función chatterCallback

Inicio función principal
Declaración de variables locales:
Inicialización variable global de tiempo: begin
Inicialización variables empleadas para copiar el contenido de map.txt
line, mapfile: Variables auxiliares para la lectura del archivo map.txt
myID: Identificador del nodo móvil.

Apertura de map.txt
Si se ha abierto correctamente
Mientras haya líneas en el archivo
Copiar cada elemento por fila y columna, correspondientes al nodo fijo, posición inicial y si el nodo está completo.
Fin de mientras
Fin del si
Si no se ha abierto correctamente
Imprimir mensaje de error.
Fin del sino
Cierre de map.txt
Cálculo del tiempo T necesario para calcular todas las medidas. (comentado en detalles técnicos).
Inicialización del nodo móvil por puerto serie (Par sensor-Raspberry Pi).

```
Si hay error
    Imprimir mensaje de error.
Fin del si
Lectura del nodo detectado.
Si devuelve -1: Error al enviar comando.
Si devuelve -5: Error al encontrarse el buffer vacío.
Si devuelve 0: Lectura del identificador del nodo realizada con éxito.
Si devuelve cualquier otra cosa: Error desconocido.
Copia identificador en variable myID.
Impresión: Nodo móvil myID inicializado correctamente.
Creación variables temporales.
Cambio de nombre al archivo logfile_name por la fecha y hora actual.
Creación del nodo mobile.
Definición como publicador en el topic 'Upstream'.
Creación de vector de subscriptores.
Definición del nodo mobile como subscriptor del topic 'Downstream'. Cuando
llega un mensaje a este topic, entra la función auxiliar chatterCallback.
Definición de la frecuencia de repetición del bucle.
Mientras no haya fallos en ROS
    Declaración variable local de tipo activar 'call'
    Inicialización del campo max_range de call.
    Declaración de variable auxiliar para guardar la medida.
    Modificación del nombre del archivo a abrir incluyendo la ruta de ubi-
    cación.
    Para i=0 hasta que se llegue al número máximo de nodos conectados.
        Mide distancia entre el nodo móvil y nodo fijo i-ésimo de mapa.txt.
        Abre archivo outputme.
        Copia tiempo, nodo origen (nodo móvil), nodo destino (nodo fijo i)
        y medida entre ambos.
        Cierra archivo outputme.
        Si el nodo fijo leído está completo.
            Copiar identificador del nodo en campo id de call.
            Talker publica mensaje call.
            Pausa de T: Necesaria para que el nodo subscriptor correspon-
            diente al topic 'Upstream' pueda recibir el mensaje.
        Fin del si
        Activación de posible recepción de mensajes.
    Fin del para
    Pausa para completar la frecuencia de repetición predefinida.
```

Fin del mientras
Fin función principal

Pseudocódigo *static*

Declaración de librerías:
ROS: Incluye todos los elementos para programar en C++ las diferentes herramientas de ROS.
Mensajes activar/measure: Mensajes personalizados, necesarias para poder utilizar los mensajes.
Cntronmod: Creación de conexión puerto serie con sensores, inicialización de los mismos, identificación de sensores, realización de medidas.
Generales: Estándar, vectores, cadenas, ficheros.

Definición de variables globales:
Id: Identificador del nodo fijo
nodoBase: Variable asociada al nuevo nodo identificado.
chatter_pub_back: Variable puntero tipo publicador.
map, pi, x, y, z: Variables para almacenar la información contenida en map.txt

Inicio función chatterCallback: Esta función se activa cuando se publica en el topic 'Upstream' un mensaje de tipo activar procedente del nodo mobile.
Definición variable de medida auxiliar.
Si la identificación recibida en el mensaje call es igual a la identificación obtenida por la función getBaseID().
Definición de variable tipo measure: mea.
Para i=0 hasta que se llegue al número máximo de nodos conectados.
Si el identificador del nodo encontrado es distinto al del nodo actual.
Modifica los tres campos de la variable 'mea' incluye nodo origen, nodo destino y medida entre ellos.
Fin del si
Fin del para
Medida de distancia entre el nodo actual y el nodo móvil.
Actualización de variable mea.
Publicación del mensaje en el topic 'Downstream'.
Fin función chatterCallback

Inicio función principal
Declaración de variables locales:

```
Inicialización variables empleadas para copiar el contenido de map.txt
line, mapfile: Variables auxiliares para la lectura del archivo map.txt
Apertura de map.txt
Si se ha abierto correctamente
    Mientras haya líneas en el archivo
        Copiar cada elemento por fila y columna, correspondientes al nodo
        fijo, posición inicial y si el nodo está completo.
    Fin de mientras
Fin del si
Si no se ha abierto correctamente
    Imprimir mensaje de error.
Fin del sino
Cierre de map.txt
Inicialización del nodo fijo por puerto serie (Par sensor-Raspberry Pi).
Si hay error
    Imprimir mensaje de error.
Fin del si
Lectura del nodo detectado.
Si devuelve -1: Error al enviar comando.
Si devuelve -5: Error al encontrarse el buffer vacío.
Si devuelve 0: Lectura del identificador del nodo realizada con éxito.
Si devuelve cualquier otra cosa: Error desconocido.
Copia identificador en variable global id.
Impresión: Nodo fijo id inicializado correctamente.
Modificación del nombre del nodo static en función de su id.
Definición del nodo subscriptor como subscriptor del topic 'Upstream'. Cuando
llega un mensaje a este topic, entra la función auxiliar chatterCallback.
Creación de vector de publicadores en el topic 'Downstream' mensajes de tipo
measure.
Activación de posible recepción de mensajes.
Fin función principal
```

Varios detalles pueden ser destacados de la implementación de ambos nodos:

- Como se comentó, se han conseguido crear vectores de nodos de dimensión variable. Esto permite ahorrar espacio de memoria y así mismo proporciona el factor de escalabilidad que se iba buscando. Simplemente partiendo del fichero *mapa.txt* (ver figura 5.4), se extrae toda la información necesaria sobre el número de nodos conectados y demás características.

- Existen limitaciones en el tamaño de la red de sensores, esto es debido a que el *router WiFi*

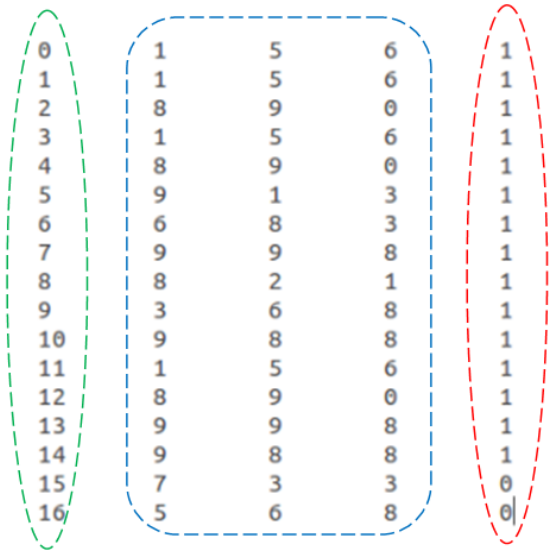

Figura 5.4: De izquierda a derecha: en verde identificador del nodo, en azul posición previa del nodo no necesaria a priori, variable que indica si el nodo está completo. 1=SI, 0=NO

tiene una capacidad máxima de dispositivos conectados. Por ello, existe la posibilidad de que los nodos sean parciales (cualidad identificada en la última columna de *mapa.txt*), es decir, solamente estén compuestos por el sensor. En ese caso el envío de medidas no sería viable. Sin embargo, el nodo móvil si podría medir y enviar dicha medida.

– A la hora de obtener las distancias a las *landmarks*, en SLAM es importante que la correlación entre diferentes medidas de un mismo punto sea alta ya que esto proporciona fiabilidad en la medición. El cálculo de la posición relativa del robot móvil a partir del mapa generado va a ser más fiable ya que, como se puede observar, la medida de distancia entre el nodo móvil y los fijos se realiza continuamente por duplicado, por parte de dicho nodo y posteriormente por parte de los nodos fijos.

5.3.5. Configuración *Cmakefile.txt*

Para completar un correcto funcionamiento, es necesario que el archivo *CMakefile.txt* sea modificado de la siguiente forma. En primer lugar, para que ROS sepa de la existencia de los nuevos archivos *.cpp* que constituyen los nodos *mobile* y *static*, estas líneas deben ser añadidas:

```
rosbuild_add_executable(static src/static.cpp
src/cntronmod.cpp src/cntronbase.cpp)

rosbuild_add_executable(mobile src/mobile.cpp
src/cntronmod.cpp src/cntronbase.cpp)
```

Como se puede observar, además de añadir los dos nodos creados, se incluyen los archivos *ctronmod.cpp* y *ctronbase.cpp*, los cuales son necesarios para poder comunicarse con los sensores, como se vio en el punto anterior.

Por otro lado, para que ROS sepa que dos nuevos mensajes personalizados han sido creados, debe descomentarse la siguiente línea:

```
rosbuild_genmsg()
```

Todas estas modificaciones son imprescindibles para poder compilar el proyecto y poder ejecutarlo posteriormente con los comandos:

```
$ rosmake
$ rosrun interbeacon mobile/static
```

5.3.6. Configuración de la red de sensores inalámbrica

Hasta ahora, el concepto de red de sensores no ha sido necesario abordarlo de forma práctica. Esto es debido a que, inicialmente, las diferentes pruebas fueron realizadas en un ordenador local, donde los nodos fueron creados de forma virtual. Pero llegados a este punto, la extrapolación a máquinas reales e independientes es necesaria para poder constituir la red de sensores. Se comentó que ROS tenía diferentes herramientas que facilitaban mucho la creación y mantenimiento de comunicación entre varias computadoras remotas.

En apartados anteriores se realizó toda la configuración de direcciones IP y *hostnames* de cada una de las *Raspberry Pi* que van a permitir, junto a los sensores, formar la red de sensores. Es por ello que el siguiente paso consiste en dejar claro quién es quién dentro de ROS para poder conseguir comunicación multidireccional.

En cualquier aplicación creada en ROS, el primer paso es poner en funcionamiento el servicio principal de este sistema, conocido como *Roscore*. En el caso de tener múltiples máquinas remotas en una misma red, solamente una de ellas debe ejecutar *Roscore* y el resto debe saber dicha información de alguna manera. Se consideró oportuno que el nodo móvil activara este servicio.

Para forzar a una máquina la ejecución de *Roscore*, hay que abrir un nuevo terminal e introducir:

```
$ export ROS_MASTER_URI=http://192.168.1.100:11311
(dirección IP del nodo móvil y puerto de comunicación)
```

A continuación, hay que decir al sistema quién eres introduciendo:

```
$ export ROS_IP=192.168.1.100
```

Además, para asegurar la eficacia de las conexiones, es necesario decirle al sistema que se va a emplear SSH:

```
$ export ROSLAUNCH_SSH_UNKNOWN=1
```

Finalmente, se ejecuta el comando que acciona roscore:

```
$ roscore
```

Así mismo, si se abre un nuevo terminal, este nodo físico ya está en disposición de ejecutar el nodo *mobile*, pero es conveniente esperar a que el resto de nodos estén configurados para evitar problemas de comunicación.

Una vez puesto en marcha el nodo móvil, se va a desarrollar la explicación de configuración de los nodos fijos que va a ser prácticamente común para todos ellos salvo por mínimas modificaciones.

En primer lugar, al ya existir un *roscore* activo y al querer que todos los nodos estén comunicados en una misma red, debe ejecutarse lo siguiente:

```
$ export ROS_MASTER_URI=http://192.168.1.100:11311
  (dirección IP del nodo móvil y puerto de comunicación)
  (Línea común para todos los nodos)
```

Al igual que antes, hay que decir a ROS qué nodo eres para que el resto pueda identificarte:

```
$ export ROS_IP=192.168.1.101
  (Esta dirección está preconfigurada en el router WiFi y
  varía según el nodo)
```

Simplemente quedaría por ejecutar el nodo *static* en cada nodo físico y la red estaría finalmente establecida.

Todo esto explicado debe realizarse cada vez que el sistema se reinicie, y es debido a que la cantidad de computadoras remotas es variables, se consideró oportuno automatizarlo. Esto se verá en el último apartado de este capítulo.

5.3.7. Herramienta *rqt*

Esta herramienta gráfica ha sido de gran importancia para depurar errores en las diferentes pruebas. De una forma visual se puede comprobar qué nodos están conectados y cómo están

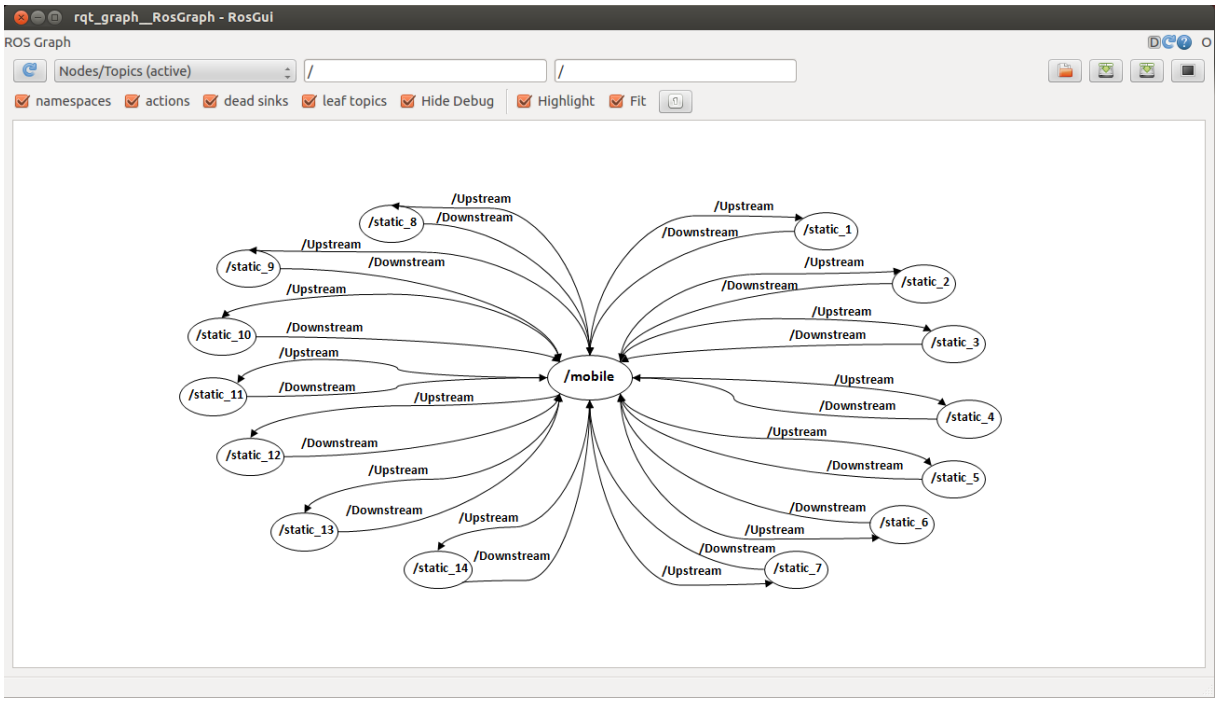

Figura 5.5: Grafo de comunicación entre nodos, empleando herramienta *rqt*

interaccionando entre ellos. En la figura 5.5 se puede observar la representación que *rqt* proporciona.

Parece claro el hecho de identificar errores ya que la consecuencia de los mismos sería la inexistencia de uno de los nodos o de una de las líneas que representan la transmisión de mensajes a través de los '*topics*' *'Upstream'* y *'Downstream'*.

Capítulo 6

Experimentos

6.1. Introducción

A la hora de realizar pruebas en un nuevo sistema recién establecido, es importante seguir una serie de pasos que van a permitir localizar posibles errores de manera más eficiente, aislarlo y darle solución.

Por este motivo, a lo largo de este capítulo se va a comentar la estrategia seguida a la hora de poner en funcionamiento el sistema creado. Hay que tener en cuenta numerosos factores, desde el simple hecho de tener todas las fuentes de alimentación disponibles, hasta lograr el establecimiento correcto y ordenado de la red inalámbrica de sensores.

Posteriormente se explicarán una serie de problemas que se han ido presentando a lo largo de los experimentos, tales como la cantidad de dispositivos conectados a la red o la necesidad de automatizar el sistema.

El factor de hacer el sistema lo más automático posible se va a convertir en algo prácticamente imprescindible debido a que el sistema tiene una dimensión variable por lo que parecería absurdo pensar en realizarlo todo manualmente cuando se disponga, por ejemplo, de 50 nodos. Como solución a la automatización se proponen la creación de una serie de scripts que irán concatenando órdenes a medida que avanza el proceso.

Finalmente se comentarán una serie de datos asociados a los resultados obtenidos en diferentes pruebas.

6.2. Puesta en marcha del sistema

6.2.1. Inicialización

Todo sistema necesita realizar una serie de tareas ordenadas para conseguir establecer la puesta en marcha del mismo de forma adecuada.

Partiendo de que la finalidad es crear una red sensorial de N nodos, siendo N el número variable de nodos, interconectados entre sí, capaces de realizar mediciones de distancia entre ellos, y transmitirlas, los pasos seguidos son los siguientes:

- Activación del *router WiFi* y de los N nodos que componen el sistema.
- Una vez comprobado los nodos conectados, los programas deben ser cargados a cada una de las *Raspberry Pi* empleando el protocolo SSH. Posteriormente deben ser compilados para su correcto funcionamiento.
- En el nodo móvil, aquel donde se ha decidido ejecutar el proceso principal *RosCore*, debe realizar las configuraciones ya explicadas en el capítulo 5. Así mismo, cada una de las N *Raspberry Pi*, debe ejecutar su configuración de red cada vez que se quiera formar el sistema.
- A continuación, mediante SSH cada uno de los programas debe ser ejecutado, comenzando por los nodos fijos y siguiendo el nodo móvil.

Hecho esto el sistema queda configurado y ejecutándose correctamente, generándose el archivo con los resultados de las medidas cada vez que se ejecuten los programas.

6.2.2. Problemas

Como se comentó en anteriores capítulos, el sistema se fue montando poco a poco. Se comenzó con un nodo móvil y dos nodos fijos. Posteriormente la red se aumentó a diez nodos para finalmente llegar a la veintena. Los dos primeros casos se establecieron exitosamente, pero a la hora de realizar la última ampliación es cuando se detecta el primer problema.

Como método de comprobación de los dispositivos conectados correctamente a la red WiFi creada, se empleó la misma herramienta que se usó anteriormente para establecer las direcciones IP de cada nodo. En este caso el objetivo era confirmar los dispositivos conectados observando el estado de la red.

El número de dispositivos conectados no superaba los dieciséis. En primer lugar se pensó que quizás, al haber conectado todo de una vez sin hacerlo de manera secuencial y escalada, el sistema había colapsado. Por ello se reinició todo pero, esta vez, los nodos se iban conectando uno

a uno activando el sucesivo una vez se verificaba la conectividad del actual. Desafortunadamente, se llegó de nuevo al mismo límite.

Normalmente, la gran parte de los *router WiFi* tienen acotado el número de dispositivos a los que dar conexión. Teóricamente este número va desde 0 a 255, pero en la realidad esto no es así ya que la mayoría de estos aparatos comerciales fija un máximo que varía entre 8, 16 o 32 dispositivos. Los más avanzados permiten modificar el valor de dicho límite pero no era el caso del *ASUS DSL-N12E*, el cual únicamente daba la posibilidad de conectar 16 dispositivos en red.

Aparentemente esto puede ser un problema, pero la solución al mismo es sencilla, bastaría con emplear un *router* más potente. En el caso en cuestión, se añadieron el resto de nodos pero de forma parcial, es decir, dotados únicamente de capacidad de medición y no de computación y envío de mensajes. De esta manera, aunque la posterior estimación de la posición de estos nodos en particular iba a ser menos precisa, el cálculo estimado de la localización del nodo móvil iba a ser más correcta debido a la adquisición de más datos.

Por otro lado se llega a la conclusión de que automatizar el sistema de alguna forma iba a ser una necesidad realmente importante. La automatización del sistema fue un concepto que se ha ido tratando anteriormente ante la posibilidad de un sistema de un número muy elevado de nodos. Por otro lado, en los diferentes códigos se trató de generalizar, dando la posibilidad de escalar el sistema.

Sin embargo, para completar dicha autonomía se pensó que las tareas de puesta en marcha del sistema iban a ser primordiales ya que, por ejemplo, acceder manualmente a las 16 *Raspberry Pi* para ejecutar los programas pertinentes o configurar la comunicación sería una tarea la cual conllevaría un coste temporal que claramente desemboca en una ineficiencia total del sistema.

ROS proporciona una utilidad diseñada para resolver este tipo de problemas pero, como alternativa más dinámica y adaptable se pensó en la creación de diferentes *scripts bash* los cuales iban a permitir realizar todas las tareas de manera prácticamente simultánea entre cada una de las *Raspberry Pi* solucionando así el problema anteriormente planteado.

6.2.3. *Scripts*

Como solución versátil al problema de automatización se realizaron una serie de *scripts*. Inicialmente se quiso dar solución simplemente a la configuración y ejecución de los programas pertinentes de forma remota, para a continuación resolver problemas algo más sencillos pero realmente útiles.

En la figura 6.1 se puede observar un esquema de la configuración general del sistema.

Se parte de *finalMainStartUp* el cual permite ejecutar a su vez roscoreremotestartup, staticremotestartup y mobileremotestartup de forma paralela en tres ventanas independientes en el PC local. De esta forma se puede observar la ejecución de los mismos por si algún fallo surgiese.

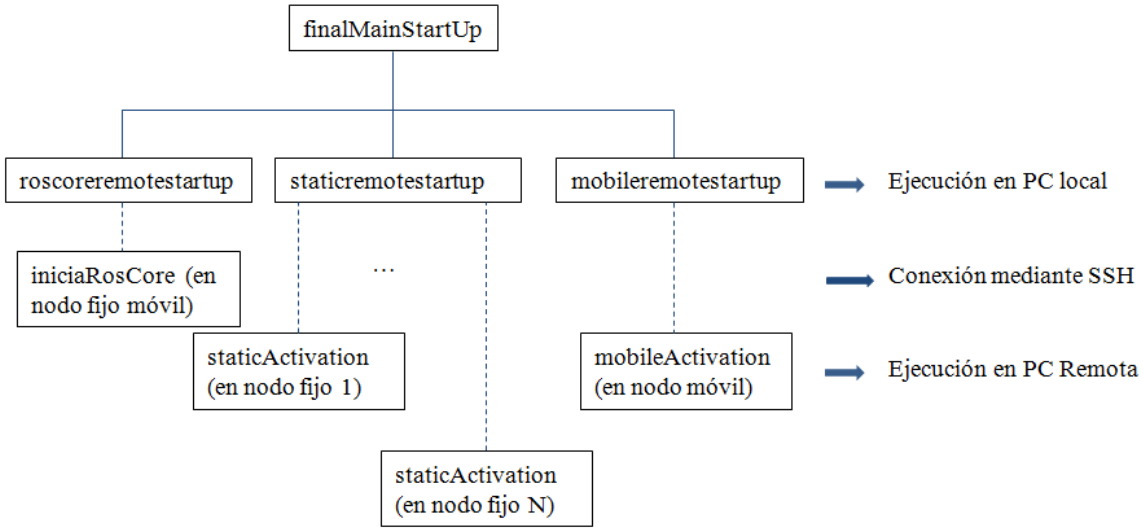

Figura 6.1: Esquema inicialización del sistema mediante *scripts*

A continuación, *roscoreremotestartup* conecta con el nodo móvil vía SSH y ejecuta de forma remota el *script iniciaRosCore* el cual configura la red y ejecuta el proceso *RosCore* también de manera remota.

Seguidamente *staticremotestartup* conecta con las N *Raspberry Pi* mediante SSH para ejecutar a continuación el *script staticActivation* el cual es diferente en cada nodo fijo ya que las direcciones IP de los mismos cambian y una de las tareas de este *script* es configurar el estado de red de cada nodo fijo. Así mismo, ejecuta el nodo *static* en cada uno de los dispositivos, quedando a la espera del aviso del nodo móvil.

Finalmente *mobileremotestartup* será el encargado de conectarse vía SSH con el nodo móvil y ejecutar *mobileActivation* el cual tiene diferentes tareas que son realizar la configuración de red pertinente y ejecutar remotamente el nodo *mobile*.

En ese momento el sistema queda establecido y ejecutándose. Si se desea ampliar el número de nodos fijos, los correspondientes scripts deben ser modificados, pero dichos cambios son insignificantes comparando con las posibilidades que proporciona el sistema automatizado.

Además de los mencionados, se crearon los siguientes *scripts* auxiliares que van a permitir una variedad de funciones realmente interesantes:

- *get_logs*: Copia el archivo, generado tras la finalización del proceso completo en el nodo móvil, y copiarlo al PC local.

- *raspberry_halt*: Apaga cada una de las *Raspberry Pi* de forma ordenada y controlada para no estropear dichos los dispositivos.

– *share_map*: Copia el archivo *map.txt* a cada nodo fijo de forma remota.

– *share_src*: Copia los códigos fuentes a cada una de las *Raspberry Pi* y los compila remotamente.

– *stop_all*: Finaliza todos los procesos relacionados con ROS para reiniciar el sistema sin conflictos entre los nuevos procesos y los antiguos, mediante el comando *kill*.

6.3. Resultados obtenidos

6.3.1. Registro de datos

Tras realizar las diferentes pruebas, la forma del archivo generado en el nodo *'mobile'* a partir de las medidas se puede ver en la figura 6.2:

log_2014_12_18_11_54.txt

6.649856	11	7	3.390000
6.649871	11	8	0.000000
6.649887	11	9	0.000000
6.649903	11	10	1.420000
6.649920	11	12	1.200000
6.649936	11	15	1.620000
6.649951	11	13	1.300000
6.649969	11	255	1.430000
6.660418	255	12	1.690000
6.910443	255	15	2.090000
7.147279	12	1	1.480000
7.147338	12	2	3.410000
7.147358	12	3	3.550000
7.147376	12	4	4.770000
7.147393	12	5	0.200000
7.147435	12	6	0.590000
7.147454	12	7	0.450000
7.147471	12	8	3.740000
7.147487	12	9	3.090000
7.147503	12	10	0.660000
7.147533	12	11	0.900000
7.147555	12	15	0.000000
7.147573	12	20	0.530000
7.147590	12	255	1.500000
7.148866	15	1	3.290000
7.148889	15	2	2.470000
7.148909	15	3	2.570000
7.148926	15	4	1.630000
7.148943	15	5	1.020000
7.148959	15	6	2.660000
7.148975	15	7	0.380000
7.148995	15	8	4.860000
7.149011	15	9	2.530000
7.149030	15	10	1.750000

Texto plano ▾ Ancho de la tabulación: 8 ▾

log_2014_12_18_11_54.txt

7.149030	15	10	1.750000
7.149137	15	11	1.660000
7.149161	15	12	0.200000
7.149180	15	13	1.620000
7.149197	15	255	2.000000
7.160884	255	13	0.600000
7.411494	255	1	0.520000
7.648278	13	1	3.100000
7.648303	13	2	2.000000
7.648320	13	3	3.050000
7.648370	13	4	0.280000
7.648397	13	5	0.520000
7.648417	13	6	0.000000
7.648434	13	7	0.330000
7.648468	13	8	1.270000
7.648491	13	9	0.000000
7.648508	13	10	0.260000
7.648525	13	11	1.790000
7.648542	13	12	0.160000
7.648559	13	15	1.290000
7.648575	13	255	0.980000
7.649743	1	2	1.100000
7.649768	1	3	2.480000
7.649785	1	4	0.470000
7.649806	1	5	0.460000
7.649825	1	6	2.470000
7.649891	1	7	0.990000
7.649912	1	8	2.270000
7.649929	1	9	2.230000
7.649945	1	10	0.120000
7.649967	1	11	1.800000
7.649984	1	12	1.510000
7.650000	1	15	3.220000
7.650016	1	13	3.160000

Texto plano ▾ Ancho de la tabulación: 8 ▾

Figura 6.2: Captura de pantalla del archivo log_2014_12_18_11_54.txt

El nombre del archivo es generado automáticamente en función de la fecha y hora de su

creación. Se puede observar que las medidas se van tomando continuamente consiguiendo una precisión bastante buena. En este caso las pruebas fueron realizadas en interior, por lo que las interferencias debido a efecto rebote de las ondas de radio frecuencia con los diferentes elementos del entorno pueden causar medidas *'outlier'*. La frecuencia de muestreo va a permitir adquirir un gran número de medidas y, consecuentemente, cuando se vaya a trabajar con ellas esto va a dar la posibilidad de conseguir una convergencia más rápida.

6.3.2. Aplicación: RO-SLAM

La creación de este protocolo de comunicación entre nodos, capaces de tomar medidas de rango y que componen una red inalámbrica de sensores, fue concebida para la aplicación real de localización y creación de mapas simultáneo empleando únicamente medidas de distancia. A continuación se muestra en las figuras 6.3, 6.4 y 6.5 un ejemplo de aplicación de este protocolo. Consiste en un sistema que emplea la cooperación de un vehículo aéreo no tripulado en una red compuesta por los sensores empleados en este trabajo.

Figura 6.3: Prueba realizada en espacio abierto en la Escuela Superior de Ingenieros, Sevilla

Tanto en la figura 6.3 como 6.4, se observa como los nodos están distribuidos aleatoriamente en el entorno, estando algunos de ellos situados en altura. Todos esos nodos son los nodos fijos. Por otro lado, en la figura 6.5 se observa el nodo móvil, acoplado al robot móvil.

Los resultados obtenidos se pueden observar en la figura 6.6, para la estimación de la posición y del mapa de nodos en el plano X-Y y, en la figura 6.7, para el caso completo en tres dimensiones.

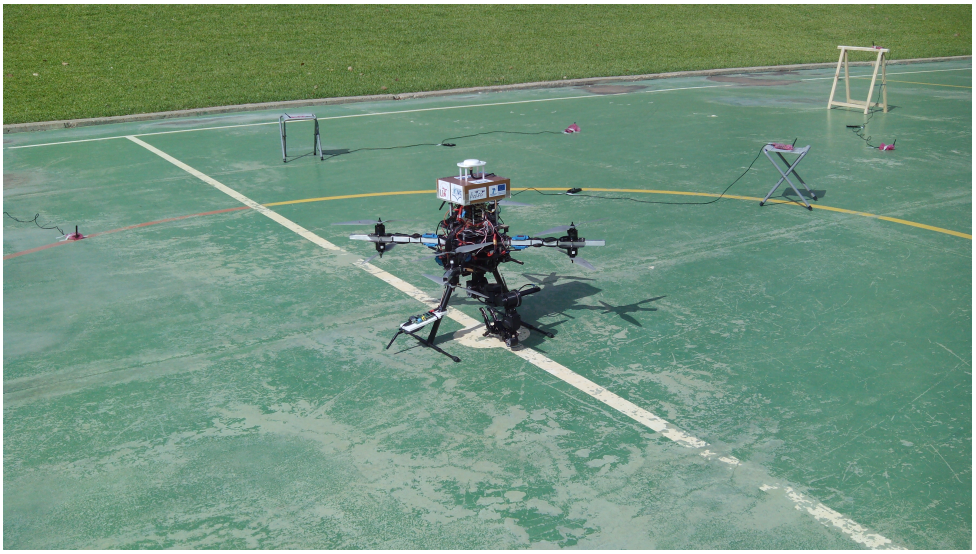

Figura 6.4: Prueba realizada en espacio abierto en la Escuela Superior de Ingenieros, Sevilla

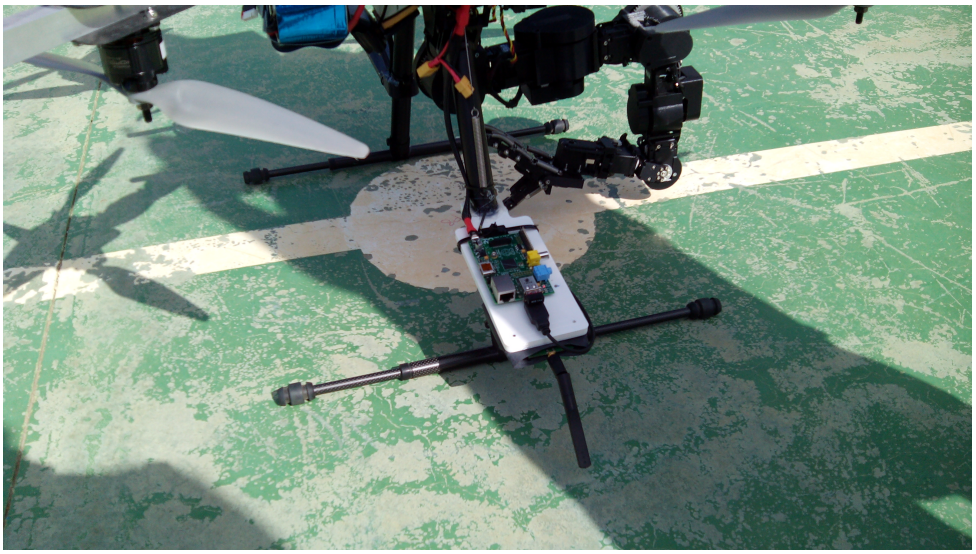

Figura 6.5: Nodo móvil acoplado al robot aéreo

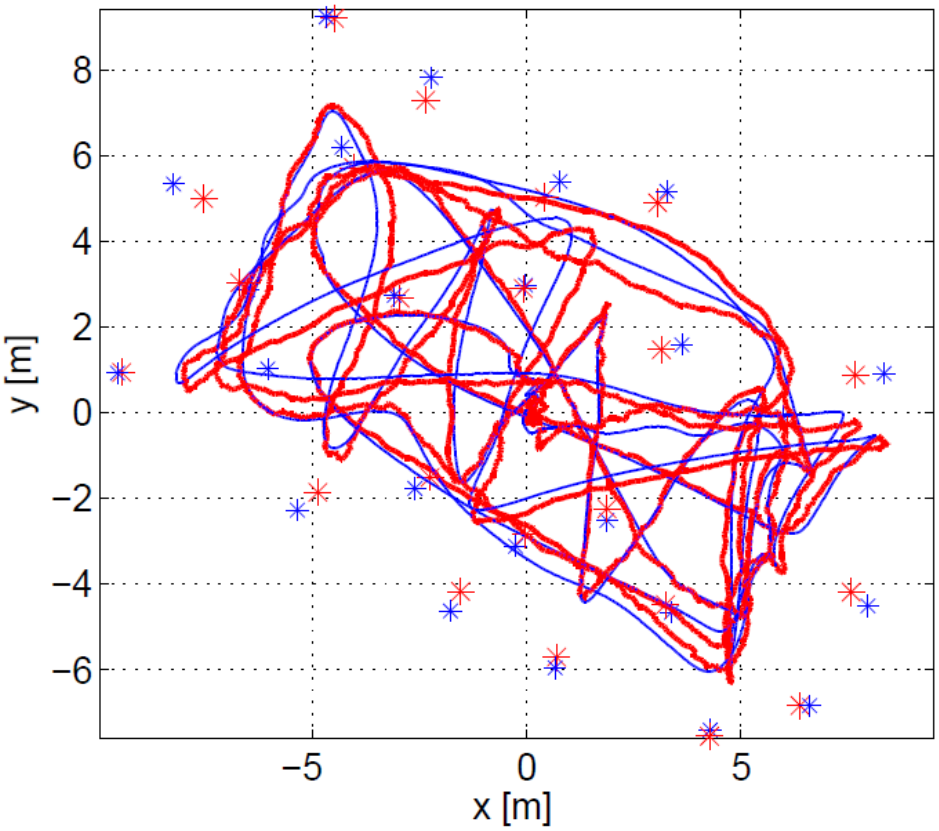

Figura 6.6: Resultados X-Y a partir de los datos recogidos

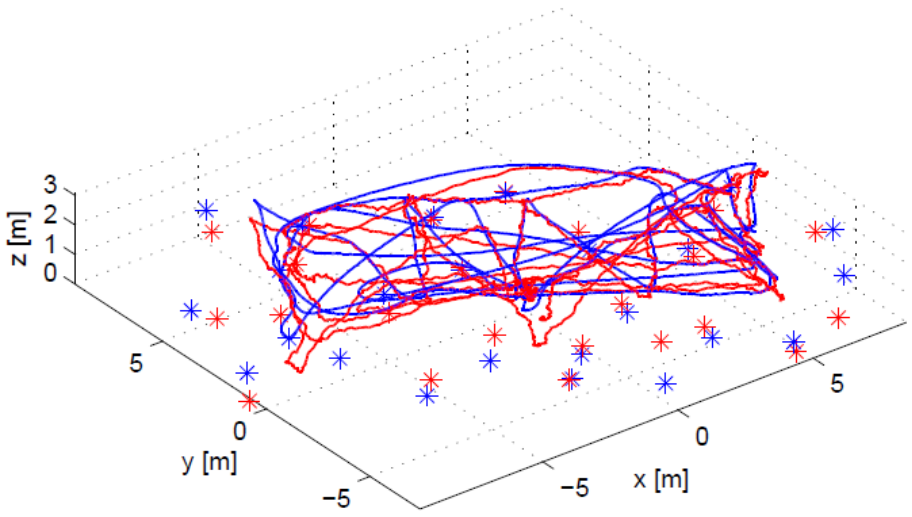

Figura 6.7: Resultados X-Y-Z a partir de los datos recogidos

Capítulo 7

Conclusiones y trabajo futuro

A lo largo de este trabajo se ha desarrollado un protocolo de comunicación entre nodos que constituían una red de sensores, llegando a conseguir buenos resultados en cuanto a escalabilidad, velocidad y a la calidad de las medidas TOF obtenidas. Como se explicó en el capítulo 2, las redes de sensores están adquiriendo protagonismo debido a su extenso campo de aplicación. Concretamente, como ejemplo de aplicación se ha visto el RO-SLAM, ver capítulo 6.

Una de las grandes mejoras respecto a trabajos previos relacionados con redes de sensores es que, ahora, se da la posibilidad de realizar experimentos *online* en vez de simular las medidas con modelos o datos. Esto se consigue ya que las medidas se van tomando constantemente, permitiendo que tanto la estimación de la posición como el mapa se vayan generando simultáneamente con la adquisición de medidas.

Por otro lado, el empleo de los sensores *Nanotron nanoPAN 5375* ha permitido dotar al sistema de medidas de distancia muy precisas como se comprueba en la figura 6.2. Esto lógicamente conlleva a que posteriores aplicaciones del algoritmo proporcionen buenos resultados, de hecho, tanto robots móviles terrestres como aéreos van a poder beneficiarse de las características de estos sensores como puede observarse en el capítulo 6.

El factor de escalado fue conseguido finalmente en la implementación, realizando un vector de nodos *'static'* de dimensión aportada por el archivo *map.txt*. Se considera que el cumplimiento de este objetivo es realmente interesante ya que las redes de sensores tienen tamaños muy diversos y, empleando el algoritmo creado, se tiene la posibilidad de adaptarlo según las necesidades del sistema independientemente del tamaño.

Pese a que los resultados obtenidos y el desarrollo del trabajo han permitido satisfacer prácticamente todos los objetivos planteados inicialmente, no todo son buenas noticias. Es por ello que, a continuación, se proponen posibles mejoras para posteriores trabajos futuros.

- *Router WiFi*: Una de las mayores limitaciones encontradas fue en cuanto al número de dispositivos conectados al *router*. Se propone el empleo de un dispositivo de mayor calidad que permita crear redes de sensores de mayor tamaño para poder hacer posteriores experimentos y hacer el factor de escalabilidad real.

- Algoritmo de toma de decisiones: Actualmente, al iniciar el sistema se podría decir que todos los nodos completos miden con todos. Mientras que esto puede resultar beneficioso en algunos casos, hay otras situaciones donde puede interesar el hecho de seleccionar ciertos nodos dentro de la red que sigan un patrón, por ejemplo, que se encuentren en posiciones con grandes obstáculos donde quizás convenga incrementar el número de medidas en esa zona.

- Mejoras de automatización: Pese a que cumplen la función de automatizar el sistema, los *scripts bash* puede que limiten un poco la velocidad de dicha automatización. Es por ello que un posible desarrollo futuro sería encontrar otra herramienta que permita realizar la misma función obtenida pero de manera más eficiente.

Bibliografía

[1] A. Torres-Gonzalez, J. R. Martinez-de Dios and A. Ollero: *Efficient Robot-Sensor Network Distributed SEIF Range-Only SLAM*, Robotics and Automation (ICRA), 2014 IEEE International Conference on, 1319-1326.

[2] L. Merino, J. Capitan and A. Ollero: *Robotica Cooperativa e Integracion con Sensores en el Ambiente. Aplicaciones en Entornos Urbanos*, 2009 Workshop Robotica09, Barcelona, Spain.

[3] Chuan-Chin Pu, Chuan-Hsian Pu, and Hoon-Jae Lee: *Indoor Location Tracking using Received Signal Strength Indicator*, Emergin Communications for Wireless Sensor Networks, 2011 intechopen.com, 230-256.

[4] A. Smith, H. Balakrishnan, M. Goraczko ans N. Priyantha. *Tracking Moving Devices with the Cricket Location System*, 2004 MobiSYS, pp.190-202, Boston, USA.

[5] F. Zhao and L. J. Guibas, 2004: *Wireless Sensor Networks: An Information Processing Approach, Elsevier: Morgan Kaufmann Series.*

[6] C. Pu, 2009. *Development of a New Collaborative Ranging Algorithm for RSSI Indoor Location Tracking in WSN*, PhD Thesis, Dongseo University, South Korea.

[7] Jiuqiang Xu, Wei Liu, Fenggao Lang, Yuanyuan Zhang, Chenglong Wang: *Distance Measurement Model Based on RSSI in WSN*, Wireless Sensor Network, 2010, 2, 606-611

[8] Steven Lanzisera, David Zats, and Kristofer S. J. Pister. *Radio Frequency Time-of-Flight Distance Measurement for Low-Cost Wireless Sensor Localization*, March 2011 IEEE SENSORS JOURNAL, VOL. 11, NO. 3, 837-845.

[9] M. Pichler, S. Schwarzer, A. Stelzer, and M. Vossiek, *Multi-channel distance measurement* with IEEE 802.15.4 (Zigbee) devices, IEEE J. Sel. Topics Signal Proc., vol. 3, no. 5, pp. 845-859, Oct. 2009.

[10] A. Torres-Gonzalez, J. R. Martinez-de Dios and A. Ollero: *An Adaptive Scheme for Robot Localization and Mapping with Dynamically Configurable Inter-Beacon Range Measurements*, Sensors 2014, 14, 7684-7710.

[11] RaspberryPi Foundation, *Raspberry Pi Hardware* [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/

[12] NanotronTechnologies, *Datasheet* [Online]. Available: http://www.nanotron.com/EN/pdf/Factsheet_nanoPAI

[13] W. Garage, *ROS tutorials.* [Online]. Available: http://wiki.ros.org/ROS/Tutorials