Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Electrónica Industrial y Automática

Desarrollo de un sistema de navegación para un robot móvil

Autor:
Blanco Abia, Cristina

Tutor:

Zalama Casanova, Eduardo
Departamento de Automática.

Valladolid, Septiembre del 2014

Índice

Capítulo 1:

Planteamiento y objetivos

1.1 Introducción y objetivos.

El trabajo que se va a realizar parte de un proyecto previo en el que se pretendía diseñar una plataforma robótica móvil de ámbito general que permitiese la investigación y el aprendizaje.

En dicho proyecto se desarrolló tanto el diseño del chasis y las características mecánicas como la selección de los sensores y actuadores, además de un controlador intermedio entre el PC y hardware del robot.

También se realizó un programa en una plataforma de desarrollo llamada Player/Stage para controlar el robot desde un ordenador. Esta plataforma tiene como función principal ocultar los detalles del control a bajo nivel de los robots. De esta manera se facilita el manejo para los operadores.

El inconveniente principal de aquel proyecto es que se hacía necesario acarrear un ordenador portátil sobre la parte superior del robot, como sistema de control de alto nivel. Esto tiene varias desventajas como una gran vulnerabilidad, ya que va fuera de la carcasa, la dificultad de manejo, el coste, etc.

Por las razones descritas se fijó como objetivo, para este proyecto, la migración del sistema de control de un ordenador a un sistema embebido en la propia plataforma. De esta manera el hardware iría más protegido, se abaratarían costes, ya que son más económicos que los ordenadores y se facilitaría el manejo.

El primer paso será estudiar los sistemas embebidos o empotrados que hay en el mercado y ver cuál de ellos se adapta mejor a las características buscadas.

Después de haber elegido una de las opciones, se deberá seleccionar el sistema operativo más acorde a los requerimientos del proyecto, la infraestructura de control y la adaptación de la misma al sistema elegido.

Otro de los objetivos de este proyecto es la incorporación de un sistema de navegación para el robot, que le permita hacer mapas del entorno en el que se encuentra para más tarde poder localizarse dentro del mismo y ser capaz de desplazarse hasta un objetivo señalado.

Para finalizar se deberán realizar pruebas experimentales sobre el funcionamiento global del prototipo desarrollado, para comprobar que se cumplen plenamente los requisitos establecidos.

Para resumir el apartado se enumerarán los objetivos descritos:

- Selección de una plataforma de control a alto nivel que solvente la problemática del anterior sistema.

- Selección del sistema operativo y de la distribución que utilizará el dispositivo de control.

- Incorporación de un sistema de navegación para que el robot se localice y sepa desplazarse esquivando los posibles obstáculos en un entorno memorizado previamente.

- Realización de todos los objetivos anteriores con un coste moderado.

1.2 Estado del arte.

Antes de comenzar con el desarrollo del proyecto se ha realizado un pequeño estudio de mercado, en el que se ha investigado qué hay desarrollado en este campo en el momento del desarrollo de este proyecto.

En primer lugar hay que tener claro qué es una plataforma de desarrollo robótico y las características que va a reunir dicha plataforma para poder compararla con las que hay actualmente en el mercado.

Una plataforma robótica es un robot que al tener características similares a los robots industriales o comerciales permite que se use como plataforma de experimentación más allá de la simulación para asegurar que los resultados reales son los esperados. Por ello estos robots se utilizan en gran medida con fines de investigación y en el desarrollo de proyectos, tanto profesional como educacionalmente.

Un robot con estos fines debe reunir unas características que lo hagan interesante para estos aspectos.

Por ejemplo debe ser capaz de desempeñar un elevado número de funciones y a su vez estar preparado para posibles ampliaciones o modificaciones de éstas en caso de ser necesarias, adaptándose con relativa facilidad a las nuevas tecnologías que vayan surgiendo.

Ya que el robot va a ser autónomo, es decir, que va a poder desenvolverse en un entorno no conocido sin intervención humana, deberá reunir las características expuestas a continuación. Estas funciones básicas son un desglose del objetivo final del proyecto, que es la navegación en un entorno interior.

- Obtener datos del entorno a través de sensores.
- Ser capaz de operar sin intervención humana.
- Desplazarse por el entorno evitando colisiones.
- Ser capaz de evitar situaciones que podrían llegar a ser peligrosas para las personas, para sí mismos o para bienes materiales.

En este caso no va a ser un robot auto mantenido ya que requerirá que una persona conecte el cargador cuando el robot esté bajo de carga aunque sea él mismo el que se desplace hasta el punto de carga.

A la hora de hacer la investigación del mercado se puede observar que es una tecnología aún poco explotada por su complejidad. La idea es que el robot debe ser absolutamente autónomo en un medio relativamente conocido.

En el mercado no es posible divisar un claro ejemplo de robot que haga las mismas funciones que deben ser desarrolladas en este proyecto. Por esta razón este trabajo es sumamente interesante.

Se podría caer en la tentación de compararlo con los llamados AGV (Automated Guided Vehicles). Este tipo de plataformas móviles están muy explotados en la actualidad pudiéndose encontrar en la industria e incluso en hospitales. Sin embargo estas plataformas utilizan un sistema de guiado diferente al tratado, tal como un raíl, una línea negra o un campo magnético, por ejemplo.

Estos sistemas se diferencian de Eusebio básicamente en que el robot va a seguir en todo momento ese camino prefijado y si encuentra un obstáculo en su camino lo único que podrá hacer es detenerse para evitarlo.

Pues bien, en este caso el robot tendrá almacenado en su memoria un mapa detallado del terreno. De esta forma se le dará una serie de coordenadas por las que tiene que ir pasando y de manera autónoma elegirá el camino que recorrer evitando los obstáculos almacenados en el mapa. Esto sucede con los nuevos ALV (Autonomous Land Vehicles) que se mueven de un punto a otro sin ayudas externas.

Sin embargo cuando el robot detecta un obstáculo, por medio de los ultrasonidos o bumpers, que no estaba definido en el mapa previamente almacenado se detendrá, al contrario que los ALV.

Por todo ello está a medio camino entre el AGV y el ALV tendiendo más hacia este último.

En cuanto al mercado de los AGV es muy extenso, pudiendo variar el precio desde los 20.000 o 30.000 dólares hasta los 200.000. En todo caso el precio es muy superior ofreciendo unas funcionalidades muy limitadas que no son las buscadas.

En cuanto a los ALV no es posible hacer un estudio de mercado debido a que

no existe un mercado como tal. Han aparecido en los últimos tiempos como proyectos o prototipos, como lo es el ya conocido automóvil autónomo de Google® (figura 1.1) y de otras corporaciones. Al ser un prototipo es muy difícil valorar su coste pero sin lugar a duda, superará el del robot estudiado.

Figura 1.1. Automóvil autónomo de Google.

Además en el año 1985 el DARPA (Defense Advanced Research Projects Agency) de Estados Unidos también inició un proyecto que finalmente abandonó para desarrollar vehículos blindados autónomos. EL prototipo que se desarrolló puede verse en la figura 1.2.

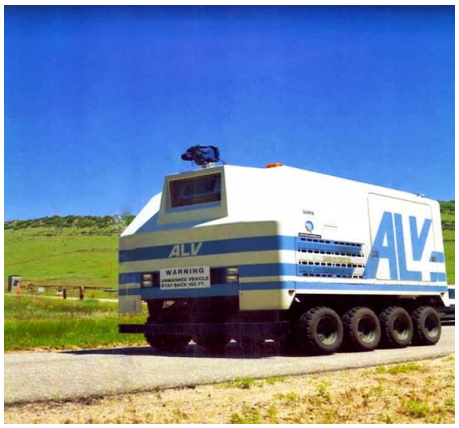


Figura 1.2. Proyecto DARPA.

Otra característica interesante del robot es que también puede ser utilizado para realizar el mapeo de la zona deseada. Se va moviendo el robot por la zona y éste la irá reconociendo por medio de los datos del láser y la memorizará para poder utilizarla posteriormente.

Este mapeo es utilizado en la actualidad por los robots limpiadores para el hogar más avanzados y su precio ronda los 500 €. Sin embargo la resolución de mapeo es sumamente inferior a la lograda con el láser utilizado y por tanto la navegación será menos segura y fiable.

1.3 Estructura de la memoria.

Una vez claros los objetivos de este proyecto se puede estructurar la memoria en capítulos, ya que ésta se compondrá de partes bien diferenciadas.

En el primer capítulo se ha realizado una definición detallada de las plataformas robóticas, de las funciones y características que reúnen, así como de los objetivos que deberá cumplir el robot una vez terminado el proyecto. También se ha realizado un estudio sobre los robots de este tipo que hay en el mercado.

En el segundo capítulo se realiza una descripción de la plataforma de partida. Se describe tanto el diseño mecánico, como la locomoción, como los sensores con los que cuenta desde el proyecto en la que se desarrolló, y a continuación se exponen las modificaciones que se van a realizar para mejorar las características generales de la plataforma y cumplir con los objetivos buscados.

En el tercer capítulo se elegirá una plataforma de control de alto nivel, analizando las ventajas e inconvenientes de las diferentes opciones. Una vez elegido el control se procederá a elegir el software en el que se va a desarrollar la navegación de igual manera.

El cuarto capítulo es una breve introducción a la Raspberry Pi, el sistema de alto nivel elegido. Este apartado se ha planteado como un pequeño manual de usuario para configurar los aspectos de la Raspberry Pi necesarios para el desarrollo de este proyecto.

En el quinto capítulo se hace una descripción del software elegido para desarrollar la navegación. Además también se dan unas nociones sobre la instalación en el sistema operativo elegido y unas lecciones básicas sobre el manejo del software que serán claves para el posterior funcionamiento del robot.

En el sexto capítulo se desarrollará el funcionamiento general de la navegación de la plataforma robótica para más tarde analizar la función de cada nodo que se utiliza por separado, dando una descripción más detallada de la tarea que realiza.

El séptimo capítulo se dedicará al desarrollo de un manual de usuario de la plataforma robótica que explique los pasos a seguir para manejarla. Por ejemplo se abarcan temas como los cambios de permisos para el correcto funcionamiento de los nodos o cómo ejecutar nodos en distintas máquinas ya que será necesario para la navegación del robot. También se explicará la secuencia de activación de los distintos nodos y las órdenes que pueden darse una vez ejecutados. También se hará una relación de las incidencias que han surgido a lo largo del desarrollo de este proyecto y la solución que se les ha dado.

En el octavo capítulo se desarrolla la experimentación con la plataforma. En este capítulo se van a describir las pruebas realizadas para comprobar si se han alcanzado los objetivos iniciales, así como la calibración de la odometría para ajustar el desplazamiento que calcula el robot a través de lo que marcan los sensores con el desplazamiento real.

El noveno capítulo es un estudio del coste económico que ha supuesto el proyecto, clasificando los costes y justificando a qué son debidos. Se dividirá el presupuesto en los costes que supondría construir un nuevo robot y los costes que ha supuesto este proyecto.

Por último, en el décimo capítulo se enumeran las conclusiones obtenidas de este trabajo, tanto en la documentación que ha precedido al desarrollo como en el transcurso del proyecto. También se hará mención de los aspectos en los que se podría mejorar la plataforma, y que quizá podrían ser objeto de un proyecto futuro.

Finalmente se incluyen la bibliografía consultada y los anexos.

Capítulo 2:

Estado del robot

En el proyecto que precede a éste, se realizó un estudio de las características que debía reunir la plataforma y se hizo un diseño mecánico, eléctrico, y de control, estudiando los pros y los contras de las diferentes opciones.

Por tanto este proyecto no parte desde cero, sino que es una evolución de la plataforma anterior, a la que se mejorarán algunas de las características de control y a la que se añadirán nuevas funcionalidades. Por tanto, antes de comenzar con el desarrollo del proyecto deberá realizarse un estudio exhaustivo de las características con las que cuenta la plataforma para conocer el punto de partida.

2.1 Características mecánicas.

En este apartado se hará un resumen de las opciones que se eligieron. En consecuencia se estará describiendo el estado actual del robot antes de dar paso a la ampliación de las características que son objeto de este proyecto.

2.1.1 Chasis

Se comenzará este estudio por la estructura exterior de la plataforma, que conforma el soporte del resto de elementos.

Para la elección del material con el que se construyó la estructura se tuvieron en cuenta tres requisitos fundamentales: la resistencia, el peso y el precio.

El material que mejor compromiso ofrece entre los tres elementos considerados es el aluminio, que tiene al mismo tiempo otras características que lo hacen aún más interesante como son la resistencia a la corrosión. Además es un material soldable y de fácil mecanizado por lo que se puede adaptar fácilmente al diseño de la estructura.

Finalmente la estructura de la plataforma se diseñó con una base de aluminio, así como las paredes, dispuestas en forma de hexágono, con el fin de abarcar más ángulos con los sensores. La estructura tiene una división interior que aporta mayor consistencia al chasis además de ayudar a distribuir el peso.

En la figura 2.1 se aprecia la forma de la estructura.

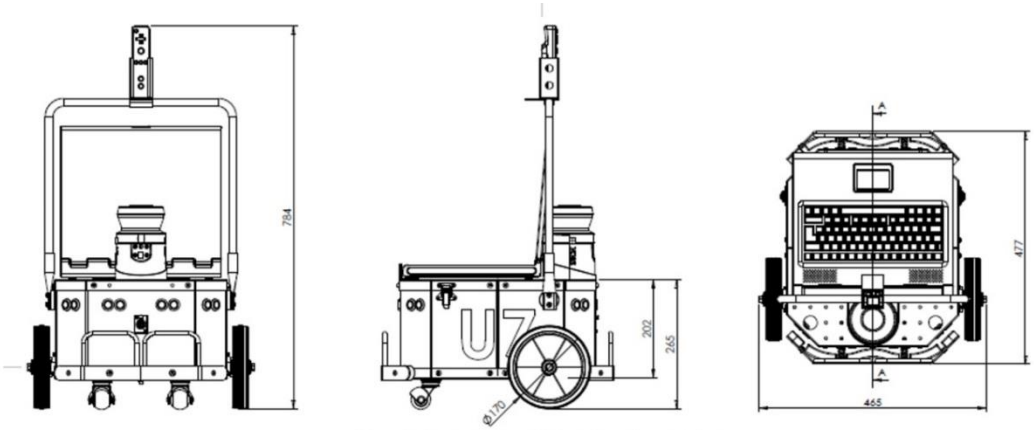


Figura 2.1. Detalle de la estructura de la plataforma.

Se puede apreciar que el diseño incluye la posición del ordenador encima de la plataforma, como sistema de control de alto nivel. Incluye además un soporte para un mando de Nintendo Wii, que en este proyecto no se va a utilizar por lo que no será necesario tampoco el soporte.

La parte superior de la plataforma es una cubierta de aluminio con una bisagra en el medio, que permite acceder a ambas divisiones del interior sin necesidad de desmontar ningún elemento.

La batería está colocada junto a los engranajes, motores y las placas necesarias para controlarlos en la parte trasera del robot. Mientras que en la mitad delantera se aloja la placa GPMRC, que transmite las órdenes de bajo nivel a los controladores de los motores. En este espacio también se guarda el dispositivo para el control manual con el que cuenta el robot.

En la imagen 2.2 se puede apreciar la distribución de los motores y la batería dentro de la parte posterior del chasis.

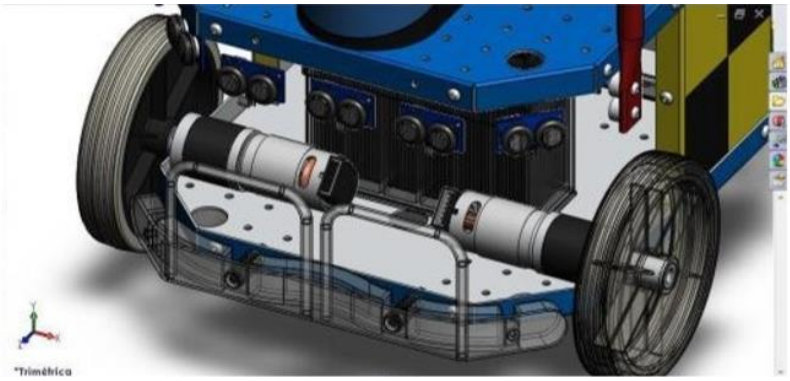


Figura 2.2. Distribución de la parte trasera.

2.1.2 Configuración cinemática.

La configuración cinemática con la que cuenta el robot es de tipo diferencial.

Esta elección se debió a que es la configuración más sencilla de todas, ya que el robot está diseñado para navegar por el interior de edificios y no va a necesitar de una configuración compleja para poder desplazarse sin problema.

La configuración diferencial se compone de dos ruedas en un mismo eje, perpendicular a la dirección en la que se moverá el robot. Para girar se introducirán diferentes velocidades en cada una de las dos ruedas, a través de su motor, de manera que se girará hacia el lado de la rueda que gire más lento.

Se puede ver un ejemplo de giro con la configuración diferencial en la figura 2.3.

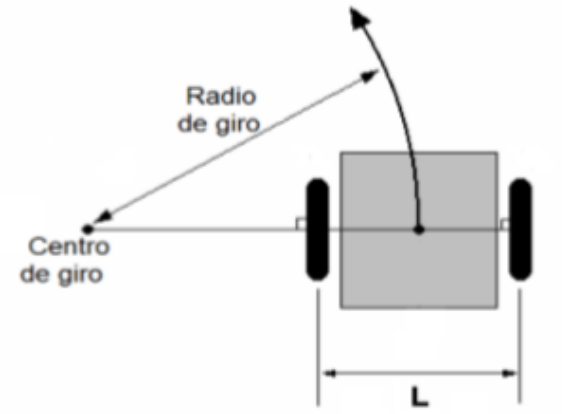


Figura 2.3. Giro en una configuración cinemática diferencial.

Para dar otro apoyo al robot y mantenerlo horizontal éste cuenta con dos ruedas locas situadas en la parte delantera del mismo y más centradas que las motrices. Con esta disposición de los apoyos se evitan posibles problemas de tracción, como el de la figura 2.4.

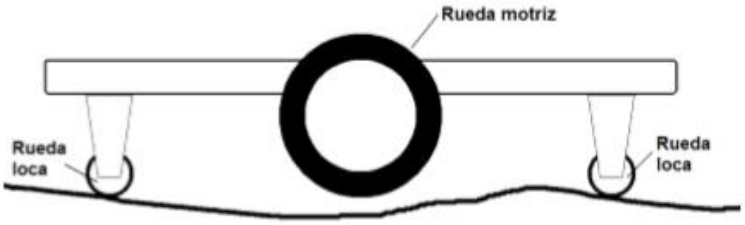


Figura 2.4. Posible problema de tracción.

La disposición de las ruedas y sus medidas son las siguientes:

- Diámetro ruedas motrices: 16 cm
- Distancia entre ruedas motrices: 42 cm
- Altura libre al suelo: 6 cm
- Distancia entre ruedas locas: 22,5 cm
- Distancia entre ejes: 24.5 cm

En la figura, la 2.5, se muestra visualmente el conjunto.

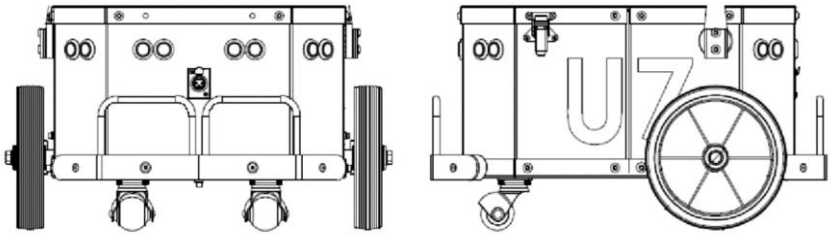


Figura 2.5. Configuración cinemática adoptada en el robot.

2.1.3 Motorización.

A continuación se van a describir las características de los motores con los que cuenta el robot, que son los que fueron elegidos en el anterior proyecto.

Estos motores de corriente continua, de la familia IG-42GM, que cuentan con una reductora de relación 1/49, cuyas características pueden apreciarse en la figura 2.6.

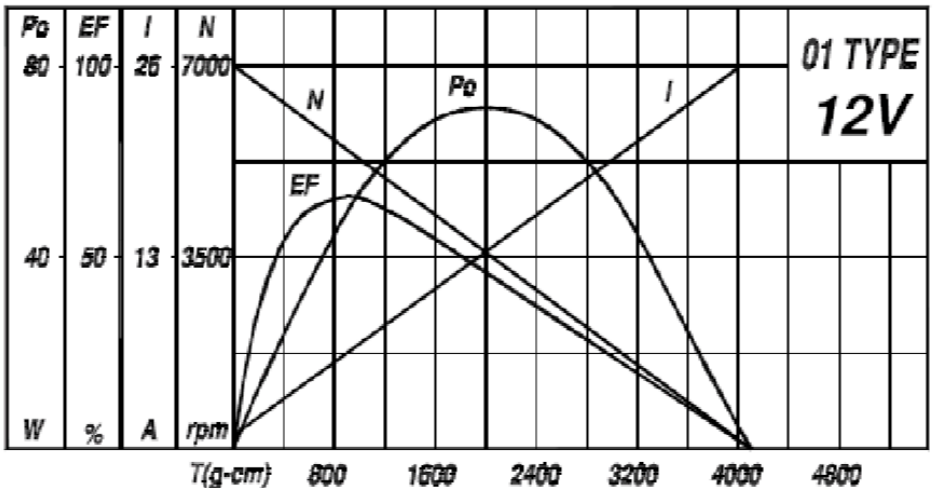


Figura 2.6. Gráfica que muestra las propiedades del motor elegido.

Estos motores son capaces de proporcionar los 15.81 Kg·cm. de par cada uno, necesarios para alcanzar la velocidad máxima deseada de 1.1 m/s, con una aceleración de 0.6 m/s y portando una carga de 20 Kg.

2.1.4 Alimentación y protecciones eléctricas.

Por último se describirán las características de la alimentación de los diferentes elementos del robot.

La fuente de alimentación con la que cuenta el robot es una batería eléctrica Yuasa NP17-12I FR, que proporciona 12 voltios de tensión y unos 15 A·h, que es capaz de proporcionar el voltaje y amperaje necesarios y por sus dimensiones encaja entre la división interior del robot y los motores.

Como formas de protección las conexiones entre los aparatos y la batería se realizan a través de una bornera de conexión (figura 2.7), evitando así cablear en exceso el interior del robot, lo que aumentaría el riesgo de enredos y cortocircuitos.

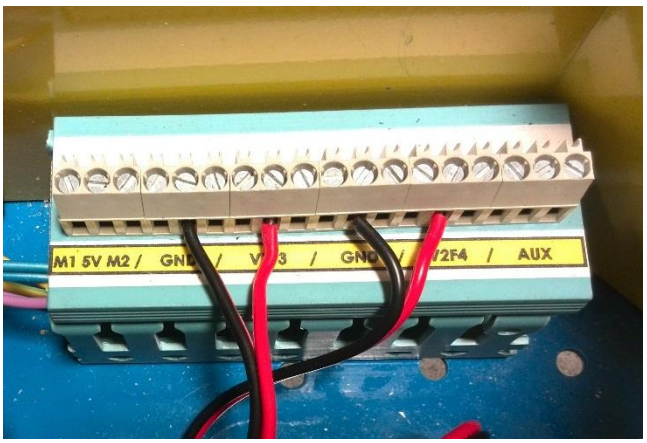


Figura 2.7. Bornera de conexión.

Además de esta medida de seguridad hay instalados fusibles para complementar la protección contra los cortocircuitos. Dichos fusibles pueden verse en detalle en la figura 2.8. Van situados intercalados entre la batería y el resto de elementos.

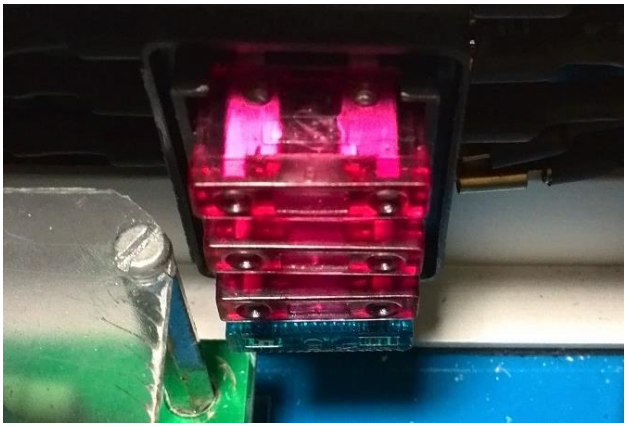


Figura 2.8. Fusibles de seguridad.

Por último hay que mencionar el sistema de carga. Se dispone de un conector situado en la parte externa del robot, con lo que es posible cargar la batería sin tener que desmontar ningún elemento. Este conector puede verse en la figura 2.9. En la siguiente imagen se aprecia además del conector, un conjunto de diodos LED que indican el estado de la batería y el interruptor general de encendido del robot.

El conector no está cableado directamente a la batería, sino que cuenta con un fusible como precaución, para proteger tanto la batería, como el cargador, como el resto de elementos en caso de defecto de la batería o uso de un cargador inadecuado.

Figura 2.9. Detalle del sistema de carga.

Para terminar con esta sección se incluye un esquema eléctrico en la figura 2.10 de las conexiones de los diferentes elementos del robot junto con las medidas de protección tomadas.

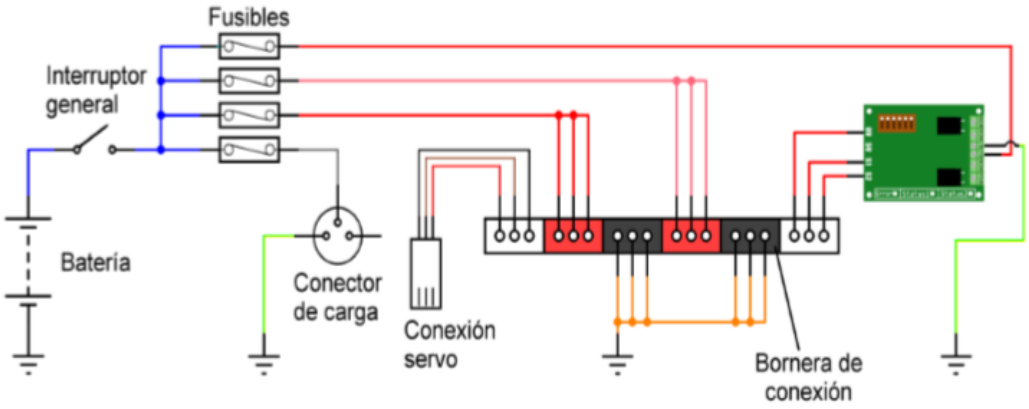


Figura 2.10. Esquema eléctrico del robot Eusebio.

2.2 Estructura de control.

Una de las características que se vieron en el primer capítulo que debía reunir una plataforma robótica es la de adaptarse fácilmente a los cambios.

Para lograr con mayor facilidad esta adaptación en la tecnología, las plataformas robóticas deben separar el “cuerpo” y la “mente” del robot. Es decir, debe hacerse una separación entre el hardware y el sistema de control a alto nivel. Tiene que haber un intermediario entre ambos para que esto sea posible, que va a ser una placa de control de bajo nivel, la cual permite al control de alto nivel manejar el hardware del robot.

La placa de control a bajo nivel tiene las funciones de recopilar la información de los sensores de la plataforma y del estado en general del hardware y de transmitirla al control de alto nivel. Al mismo tiempo sirve para procesar las órdenes que manda el dispositivo que contiene el control de alto nivel y procesarlas para obtener órdenes que puedan ser ejecutadas por el hardware del robot.

La placa de control de bajo nivel que se utiliza en este proyecto es la placa controladora GPMRC (General Purpose Mobile Robot Controller), modelo GPMRC6LC, diseñada en el centro tecnológico Cartif. Esta placa fue desarrollada en el proyecto del que parte este trabajo, sin embargo, es necesario hacer un estudio de sus funciones y su funcionamiento para poder continuar con la labor propuesta, y es el resultado de este estudio lo que se expone en este apartado.

La placa de control GPMRC está formada fundamentalmente por dos microcontroladores dsPIC30F6011, que trabajan de forma conjunta para realizar las funciones. Un microcontrolador actúa como maestro y el otro como esclavo, siendo las funciones de cada uno de los dsPIC diferentes.

Entre las funciones del dsPIC maestro se encuentran la comunicación a través de puerto serie con el dispositivo de control de alto nivel. Recopila la información del hardware del robot y la transmite. Transforma también las órdenes que recibe en forma de dos velocidades (una para cada motor) en órdenes que puede ejecutar el hardware. Por último debe detectar los errores e informar de ellos.

El PIC esclavo en cambio se encarga de captar la información que llega de los codificadores incrementales y le transmite la información al principal.

En la figura 2.11 se muestra una imagen de la placa GPMRC.

Figura 2.11. Placa GPMRC6LC.

2.2.1 Control de motores.

El control GPMRC no realiza directamente el control sobre los motores, sino que lo hace a través de dos controladores de motores, uno por cada motor, diseñados en el centro tecnológico Cartif. Cada controlador recibe señales procedentes de la GPMRC y produce una señal de salida que llega a los drivers, que gracias a la señal que reciben son capaces de regular la velocidad y sentido de los motores. En la figura 2.12 se puede ver la placa controladora de uno de los motores acoplada a su correspondiente driver.

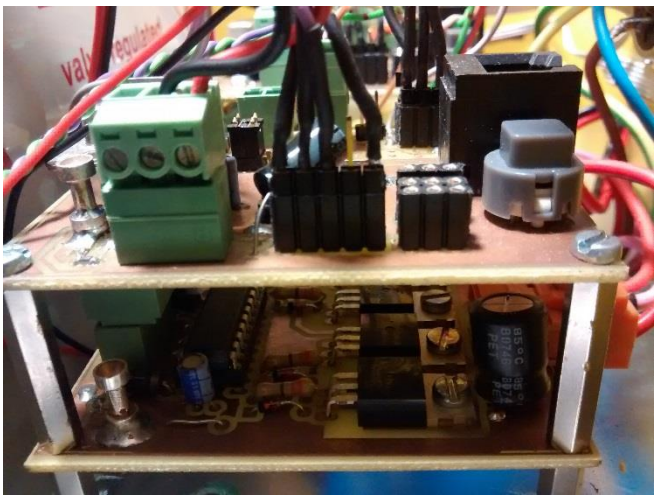


Figura 2.12. Placa controladora (arriba) y driver de motor (abajo).

2.2.2 Control de sensores.

La plataforma móvil de partida cuenta además con otra placa conectada a la GPMRC. Se trata de una placa concentradora de las señales de los sonars y los bumpers, que son respectivamente sensores de proximidad y de contacto. Más adelante se dará una definición más detallada de cada sensor. La razón de incluir esta placa controladora es que estos sensores no pueden ser controlados directamente por la GPMRC al ser la tarea de lectura muy lenta.

Esta placa consta de un microcontrolador dsPIC que toma los datos de los sensores según el orden predefinido y le pasa la información a la GPMRC a través de un bus CAN.

En la figura 2.13 se puede ver la placa concentradora de señales. En la parte superior se aprecian los dos conectores donde van conectados los sonar y los bumper (cuatro sonar y dos bumper por cada conector).

Figura 2.13. Placa concentradora de sonars y bumpers.

Por otra parte, la placa GPMRC se hace cargo de los codificadores incrementales mediante el dsPIC esclavo. A través de los pulsos de estos sensores se conoce la distancia recorrida y también la velocidad a la que se desplaza, al poder contar el tiempo entre pulso y pulso gracias a los timers o temporizadores incluidos en cada dsPIC.

2.2.3 Control manual.

Una de las funciones imprescindibles en una plataforma robótica que va a funcionar de manera autónoma es que en caso de necesidad pueda ser controlado de forma manual. Es por ello que se incluye un sistema de control manual en la plataforma.

La GPMRC incluye la posibilidad de conectar un receptor de señales de un joystick tipo Sony PlayStation® inalámbrico. Mientras que en el proyecto previo era ahí donde iba conectado, ahora se conectará el receptor directamente al dispositivo de alto nivel, que es el encargado de interpretar las señales que llegan desde el joystick.

El resto del funcionamiento de este es similar al del proyecto de partida. Para controlar el robot mediante el mando hay que mantener presionado el botón L2, que actúa como botón de "hombre muerto" y mientras se mantenga pulsado se ignorará cualquier otra orden que no sea la que envía el Joystick.

Se dirige el robot mediante el stick analógico izquierdo del mando, siendo la velocidad y la dirección proporcionales a la inclinación del stick. De esta manera solo hay que moverlo en la dirección deseada y mantenerlo en esa posición hasta que el robot llegue hasta el punto deseado, recordando mantener pulsado el botón L2.

Para la implementación de esta función tan solo es necesario un mando tipo Sony PlayStation®, que en este caso se ha utilizado inalámbrico, y un receptor USB de la señal, además del software pertinente. En la imagen 2.14 pueden verse los elementos mencionados.

Figura 2.14. Mando y receptor para el control manual del robot.

2.3 Sensores.

La plataforma robótica que fue desarrollada cuenta con unos sensores que le informan sobre su estado y sobre su entorno, de manera que pueda reaccionar de la mejor manera.

En esta apartado se enumerarán los sensores incluidos y se explicará su funcionamiento, su colocación y el uso que se le da a cada sensor.

2.3.1 Codificadores incrementales. Funcionamiento y uso. Odometría.

Este tipo de sensores van acoplados a los motores, solidarios a su eje, e informan del giro de la rueda en cada momento.

En este caso se usan dos sensores, uno para cada rueda, de efecto Hall con dos canales cada uno. Estos sensores, tienen una resolución de 20 pulsos por vuelta, pero como el motor incluye una reductora de relación 1/49 se tiene un total de 980 pulsos por vuelta del eje de la rueda.

Estos sensores son incrementales, lo que quiere decir que proporcionan el ángulo que se ha girado a partir de la posición de inicio. Con este sistema la información que se obtiene de los sensores es la dirección y velocidad del desplazamiento, de manera que se puede saber el lugar en el que se encuentra el robot solamente si el lugar de partida es conocido.

Los sensores se alimentan a 5 voltios, y cuentan con dos canales cada uno. La señal de los cuatro canales de estos sensores magnéticos los recibe el microprocesador esclavo de la GPMRC, que es el encargado de captar los pulsos que después serán procesados.

En la imagen 2.15 se muestra uno de estos sensores acoplado al motor IG-42GM.

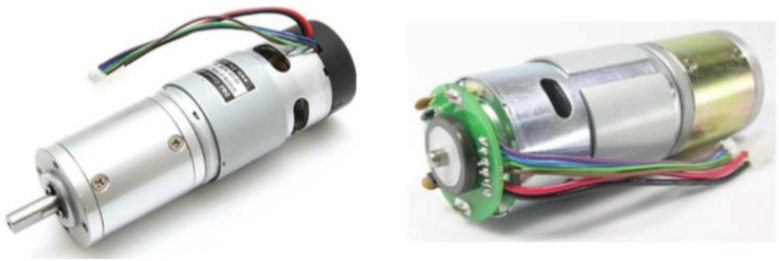


Figura 2.15. Codificador incremental acoplado al eje del motor.

Como ya se ha dicho, estos sensores permiten calcular la distancia recorrida por el robot, por lo tanto son elementos fundamentales para el cálculo de la odometría.

La odometría estudia la posición del robot durante la navegación, para ello, se basa en el número de pulsos recibidos, en el radio de las ruedas y la separación entre éstas. Con ello se determina la posición del robot a través del modelo cinemático del sistema de propulsión, que en este caso es diferencial.

La forma de estimar la posición se explicará más adelante, junto con el método de calibración de la odometría. La posición se referirá en adelante al punto medio de las ruedas motrices.

2.3.2 Sensores ultrasónicos frente a laser. Posicionamiento y detección de objetos.

La plataforma cuenta también con sensores ultrasónicos, o sonars, para poder detectar los obstáculos alrededor del robot. Como se aprecia en la figura 2.16, cuenta con 8 sensores ultrasónicos Parallax PING))), situados cuatro en la parte frontal del robot y cuatro en la parte posterior. Los dos sonars de los extremos de cada lado están situados en la cara a 45º, cubrir más área alrededor del robot.

Figura 2.16. Colocación de los sensores ultrasónicos.

Estos sonars tienen un rango de medida entre 2cm y 3m y abarcan un ángulo de entre 20º y 40º dependiendo de la forma que tenga el obstáculo. En la figura 2.17 se aprecia uno estos sensores con mayor detalle.

Figura 2.17. Sonar Parallax PING))).

A través de estos sensores puede calcularse la distancia a la que se encuentran los objetos circundantes. Sin embargo tienen un inconveniente, que no son fiables a la hora de detectar objetos cuando estos están inclinados, ya que la ráfaga de ondas ultrasónicas de las que se sirven para cumplir su cometido rebota en una dirección que el sonar no puede captar de vuelta. En la figura 2.18 puede verse un ejemplo de esta problemática.

Este es un gran problema para la navegación autónoma, ya que el robot puede ser incapaz de detectar los obstáculos que estén inclinados, por muy cerca que estén, y en consecuencia chocar con ellos.

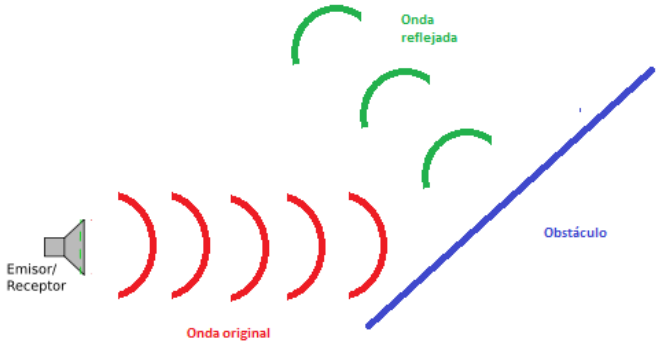


Figura 2.18. Problemática de obstáculos inclinados.

Por esta razón se ha buscado otro sistema de detección de objetos que no tenga esta problemática y que pueda ser incluido en la plataforma para cumplir de forma más robusta los objetivos del proyecto.

La solución encontrada ha sido un sistema láser, que también tendrá el emisor y receptor de la señal bajo la misma carcasa por lo que el cableado será igual de sencillo pero que sin embargo no tiene la problemática de los objetos inclinados. En este caso el objeto será detectado cuando el haz de luz es interrumpido.

El sensor elegido es el Hokuyo URG-04LX Scanning Laser Rangefinder, que detecta obstáculos desde un rango de 2 cm hasta 4 metros y efectuará un barrido del entorno en 240º, con una precisión angular de 0,36º. Operará además alimentado con 5V y cuenta con una interfaz USB y Serial con la que irá conectado al mecanismo de control. El láser elegido se muestra en la figura 2.19.

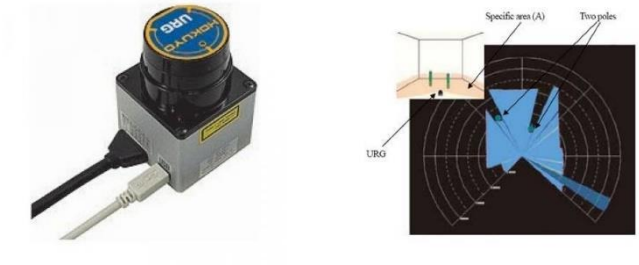


Figura 2.19. Sistema laser elegido y visualización de los datos que proporciona

En la imagen 2.20 se puede ver una imagen de la colocación del sensor en el robot.

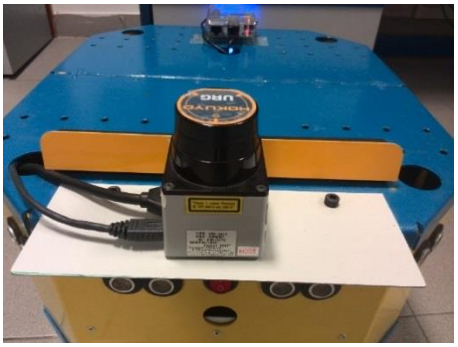


Figura 2.20. Sistema laser integrado en la plataforma robótica.

Este barrido será suficiente para poder detectar y evitar todos los objetos que haya alrededor del robot. A su vez, permitirá hacer un mapeado del entorno por el que se va desplazando para finalmente crear un mapa. En el mapa creado se localizará a sí mismo en un punto gracias a la secuencia y colocación de los objetos que detecte, y en el que se podrá fijar un objetivo al cual desplazarse.

Más adelante se detallará cómo crear un mapa a través de este sensor pero en la figura 2.21 se muestra un ejemplo de lo que se puede hacer mediante este método.

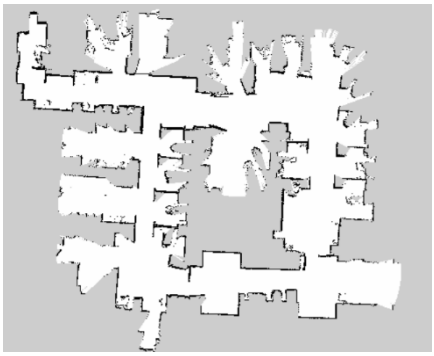


Figura 2.21. Mapa creado con el sistema laser.

Lo que finalmente se implementará es una combinación de ambos tipos de sensores para la navegación.

Con el sistema laser se creará el mapa que permita que el robot se localice y que escoja la ruta más adecuada para ir a un punto destino, y se complementará con los datos que proporcionan los ultrasonidos y los bumpers como medida de seguridad. De esta manera si por cualquier razón falla el sistema laser el robot se detendrá al acercarse un objeto a menos de 20 cm de los ultrasonidos. En último caso, si ya se ha chocado se activará alguno de los bumpers, deteniéndose de inmediato.

2.3.3 Sensores de choque. Bumpers.

Estos sensores son el último recurso contra los choques. Tienen una doble función, la primera es amortiguar el golpe contra el obstáculo que no se ha podido evitar, por un fallo en el sistema de detección de objetos o bien porque se esté manejando el robot de manera manual y se cometa un error.

La segunda función de los bumpers es de sensor, ya que están conectados a la placa concentradora de sonars e informan del choque, lo que hace que cese el movimiento. En la imagen 2.22 puede apreciarse la colocación de uno de los bumpers.

Figura 2.22. Bumper trasero en el robot Eusebio.

Hay dos bumpers, que están colocados debajo de los sonars, uno en la parte frontal y otro en la trasera del robot. Están fijados al chasis en dos puntos mediante muelles, que en caso de ser presionados activan un interruptor. De esta manera se consigue duplicar la información que llega de los bumpers, al poder saber si la colisión se ha producido de manera frontal o fronto-lateral, dependiendo de los interruptores que se activen.

En la figura 2.23 se puede ver el mecanismo que activa la colisión en detalle.

Figura 2.23. Detalle del mecanismo de los bumpers.

2.4 Arquitectura final.

Finalmente en la imagen 2.24 se muestra la arquitectura final del hardware del robot, al que habría que añadir el sistema laser y el receptor del joystick, que no se han incluido en este diagrama aclarativo porque irían conectados únicamente al sistema de control de alto nivel.

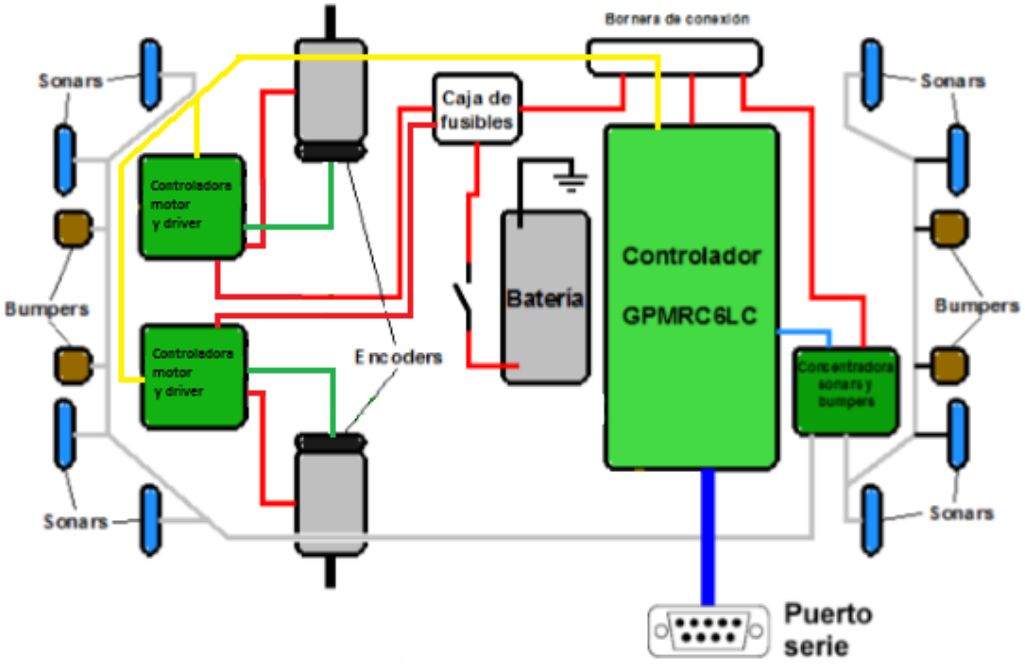


Figura 2.24. Esquema de conexiones de la arquitectura del robot.

Capítulo 3:

Control a alto nivel

En este capítulo se va a justificar la elección de los elementos que conforman el control a alto nivel del robot Eusebio.

3.1 Sistema de control a alto nivel.

Como se explicó en el capítulo introductorio, el control a alto nivel del robot se realizaba a través de un ordenador portátil.

Como ventajas se pueden encontrar su gran capacidad de procesamiento y su versatilidad a la hora de elegir sistema operativo o software de control. Por estas razones fue el método de control elegido para el proyecto anterior. Sin embargo, en este tiempo se han encontrado inconvenientes como su gran tamaño, su peso o la vulnerabilidad a la que está expuesto al encontrarse fuera de la estructura.

Estas han sido las razones para buscar otro sistema de control que se pueda incluir en el interior del robot y que deje así la parte superior libre para otros usos como el transporte de objetos.

En la Figura 3.1 se puede observar la posición del ordenador portátil, que era el sistema de control en el proyecto anterior.

Una alternativa a los ordenadores portátiles son los, cada vez más utilizados, sistemas embebidos.

Figura 3.1. Ordenador como sistema de control.

Los sistemas embebidos o empotrados son una combinación entre software y hardware que se caracteriza por que está integrado en el sistema en el que se encuentra, de manera que quede oculto a la vista. Constan de uno o varios microprocesadores digitales o microcontroladores que les permiten gobernar al sistema anfitrión del que forma parte y dotarle de 'inteligencia'.

Entre los beneficios de los sistemas empotrados se encuentran: un bajo consumo, lo que se traduce en una mayor autonomía, un menor peso que los sistemas convencionales, un precio mucho menor que por ejemplo los ordenadores portátiles, lo que los hace mucho más competitivos en el mercado, y unas dimensiones pequeñas, por lo que se pueden integrar con mayor facilidad en un sistema anfitrión.

Por estas razones, para este proyecto se considera una mejor opción un sistema embebido.

3.2 Elección de plataforma embebida.

En el apartado anterior de justificó la elección de un sistema empotrado como control de alto nivel. En este capítulo se hará un estudio de las diferentes opciones que se encuentran en el mercado en este momento, exponiendo las ventajas e inconvenientes de las más importantes. Por último se determinará la opción más favorable para este caso.

3.2.1 Arduino.

Arduino es una plataforma de hardware libre de propósito general para el desarrollo de proyectos electrónicos. Puede verse una de sus plataformas en la figura 3.2.

El hardware consiste en una placa con un microcontrolador Atmel y puertos de entrada y salida. Por otro lado el software, que también es libre, consiste en un entorno de desarrollo que implementa el lenguaje de programación propio y el cargador de arranque que es ejecutado en la placa.

Esta plataforma es pequeña, por lo que podría introducirse dentro de la estructura, y también ligera por lo que no tiene los inconvenientes de la opción anterior.

Como ventajas de esta opción se puede destacar que al ser de hardware y software libre se reduce considerablemente el precio. Además la programación en este entorno es sencilla.

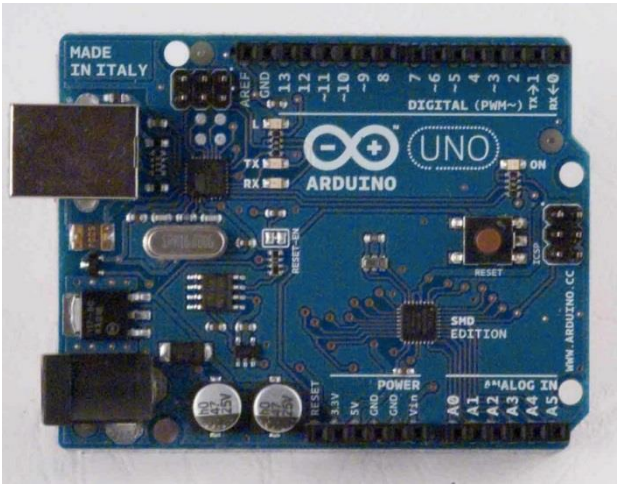


Figura 3.2. Arduino Uno.

Los inconvenientes principales de esta opción son la escasa capacidad de almacenamiento que posee Arduino, así como la dificultad para conectar todos los elementos del robot, ya que no dispone de puertos USB ni Serial. Esto lleva a considerar otras opciones.

3.2.1 Raspberry Pi.

Raspberry Pi es un ordenador de placa reducida de bajo coste desarrollado y comercializado por la fundación Raspberry Pi.

El diseño hardware contiene un procesador central (CPU) ARM1176JZF-S a 700 MHz, pudiendo hacer overclock hasta 1 GHz, un procesador gráfico (GPU) VideoCore IV, y 512 MB de memoria RAM. Utiliza una tarjeta SD para el almacenamiento permanente, que no está incluida. Esto podría parecer a priori una desventaja pero permite elegir una capacidad de almacenamiento acorde con el uso que se le va va a dar, desde un mínimo de 2GB. Tampoco incluye fuente de alimentación ni carcasa.

El hardware tiene un tamaño aproximado a la plataforma Arduino. Además, a pesar de que se trata de una plataforma comercial, su precio es muy competitivo no superando en gran medida el precio de las placas de Arduino.

Al igual que él incluye pines de entrada y salida. Sin embargo la frecuencia de reloj es mucho mayor en Raspberry Pi (hasta 1 GHz, mientras que en Arduino es de unos 16 MHz). La diferencia es más que notable y aporta mucha más velocidad de procesamiento, aunque sigue sin llegar a la de un ordenador.

La gran ventaja de esta plataforma es la variedad de sistemas operativos que se pueden utilizar y la versatilidad de los lenguajes de programación como C++ o Python.

En la figura 3.3 se muestra la plataforma Raspberry Pi.

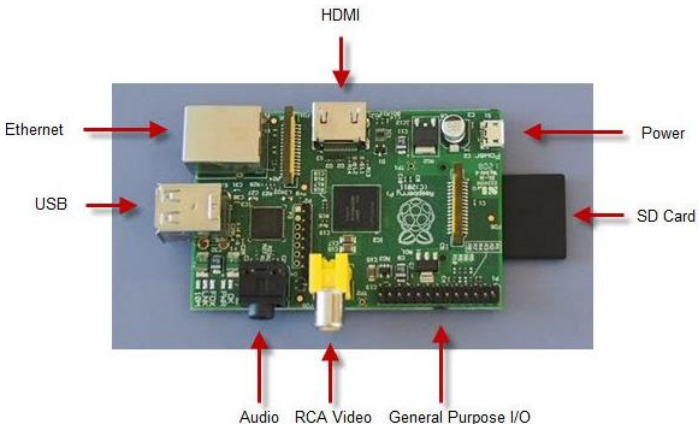


Figura 3.3. Detalle de una Raspberry Pi.

3.1.4 BeagleBone Black

Al igual que en el caso de la Raspberry, se trata de un ordenador de placa reducida, en este caso, es diseñado y desarrollado por Texas Instruments.

Texas Instruments creó las placas BeagleBoard hace años. Estas placas contaban con mejores cualidades que Raspberry pero por contra eran mucho más caras. Tiempo después decidieron lanzar BeagleBone Black con un precio mucho menor para competir directamente contra Raspberry Pi. Se pueden observar sus características muy similares y un aspecto un tanto similar al Arduino al incluir los pines hembra en sus laterales. Pueden verse sus características físicas en la figura 3.4.

Consta de una CPU ARM Cortex-A8 a 1 GHz, acelerador gráfico 3D, dos PRU (Programmable Real-Time Unit) SS RISC de 32 bits, 512 MB de RAM y 2 GB de almacenamiento interno, además de una ranura microSD. En lo que a puertos se refiere, este pequeño PC incluye una conexión USB, otra Ethernet, salida micro-HDMI y dos conectores de 46 pines.

El sistema operativo por defecto es Angstrom Linux, pudiéndose instalar también Ubuntu y Android.

Sin embargo esta plataforma tiene algunos inconvenientes significativos. El precio aunque competitivo, sigue siendo superior al de la Raspberry Pi. Además, a pesar de que cuenta con el respaldo de una gran compañía es un producto relativamente nuevo y con un recorrido bastante inferior al de la Raspberry Pi. Esto supone que hay un menor número de gente que ha trabajado con ello y por lo tanto se han desarrollado menos proyectos, pueden encontrarse menos tutoriales y los errores pueden ser más difíciles de solventar.

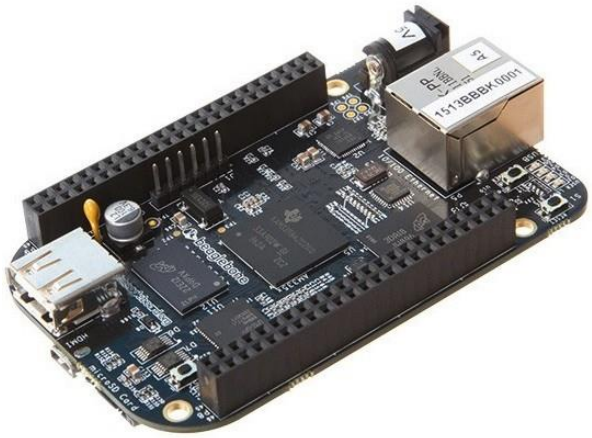


Figura 3.4. BeagleBone Black.

3.1.5 Otras opciones

Además de las grandes opciones expuestas anteriormente cabe destacar el gran número de opciones que han surgido recientemente.

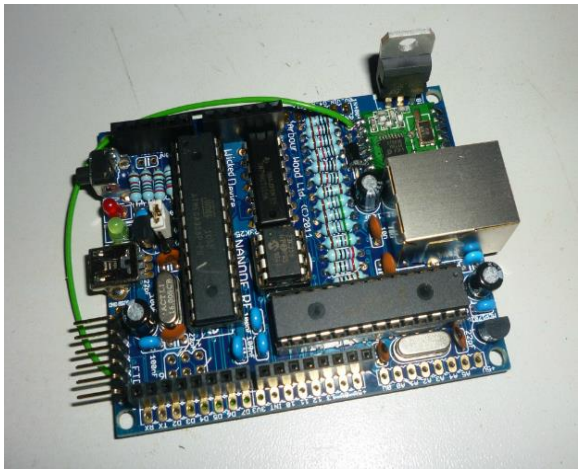


Figura 3.5. Nanode.

Un claro ejemplo de ello es la plataforma Nanode, (figura 3.5) desarrollada por un Ingeniero Electrónico de Reino Unido. Se trata de una plataforma muy similar a Arduino, pero con la ventaja de poderse conectar a Internet a través de un API (Interfaz de Programación de Aplicaciones). Se puede encontrar por un precio más económico y se puede programar desde cualquier sistema operativo a través del mismo entorno que Arduino.

Otro ejemplo, además de desarrollo español, es Libelium Waspmote (figura 3.6). Se trata de un dispositivo ideado para crear redes inalámbricas de sensores con unos requerimientos bastante específicos y destinados a ser desplegados en un escenario real. El entorno de desarrollo es el mismo que el de Arduino y el código de este se puede adaptar para poderlo utilizar. Se trata de una plataforma interesante pero destinada a otro tipo de proyectos.

Figura 3.6. Libelium Waspmote.

Además de estas dos plataformas han surgido infinidad de ellas en los últimos años que se descartan directamente por su bajo nivel de soporte.

3.1.6 Solución adoptada

A continuación se han valorado las características principales de las opciones expuestas y las conclusiones han sido las siguientes:

Se ha descartado la plataforma Arduino debido a su baja velocidad de procesamiento, la dificultad para conectarlo con el resto de los elementos del robot, como la GPMRC y las concentradoras de sonars, la escasa capacidad de almacenamiento y el lenguaje de programación.

La opción de la BeagleBone Black también ha sido descartada. Como se ha explicado es una plataforma muy similar a Raspberry Pi. Sin embargo su precio es mayor, cuenta con un puerto USB menos y su gran desventaja es que a diferencia de Raspberry Pi, es un producto relativamente nuevo, con muchos menos seguidores y desarrolladores. Se podría sin embargo justificar el uso de esta placa en el caso de que se necesitaran numerosas entradas o salidas para sensores. Sin embargo este no es el caso, ya que los sensores van conectados directamente a otras placas y todo gobernado por la GPMRC, por lo que su uso no se hace necesario.

En relación a las otras opciones vistas en el apartado anterior se deben descartar porque a pesar de que son plataformas muy interesantes, no satisfacen las necesidades de este proyecto.

La opción que se ha elegido ha sido la Raspberry Pi, ya que cuenta con grandes ventajas, como la fácil conexión al resto de elementos del robot, la conexión a internet y el gran soporte con el que cuenta. Sin embargo, como en alguna tarea se necesita gran rapidez de respuesta y por ello gran capacidad de procesamiento, se ha decidido complementar este sistema embebido con un ordenador de sobremesa con el que se comunicará por WIFI para realizar las mencionadas acciones.

Con ello se logran solucionar los problemas de espacio, peso y vulnerabilidad que suponía un ordenador portátil como sistema de control a alto nivel, a la vez que se tiene una capacidad de procesamiento suficiente para ejecutar la aplicación con la rapidez necesaria para un buen funcionamiento y libertad de elección de lenguajes de programación y sistemas operativos.

3.2 Elección del sistema operativo.

Hay seis sistemas operativos oficiales compatibles con Raspberry Pi. Dentro de estos seis, dos de ellos están pensados para actuar como centros multimedia para la reproducción de música, video etc. por lo que no son compatibles con la aplicación que se quiere dar.

Los cuatro restantes son: Raspbian, Pidora, Arch Linux y Risc OS. Todos ellos menos Risc OS están basados en el sistema Linux.

Risc OS es un proyecto de código abierto que tiene más historia que Linux. Este es un sistema operativo totalmente diferente, con un enfoque distinto, con conceptos y suposiciones subyacentes. Esto a priori no supone una dificultad pero dado que no nos es conocido este sistema, al contrario que Linux, podría repercutir en un mayor tiempo de desarrollo del proyecto por lo que se ha descartado esta distribución.

Se proseguirá ahora con las distribuciones basadas en Linux, empezando por Arch Linux. Es una distribución de Linux que está destinado a personas que ya están muy familiarizados con Linux. Es mucho más personalizable que las otras dos opciones pero hay que hacerlo todo a partir de la interfaz de línea de comandos, no en un escritorio gráfico. Dado que para esta distribución se requiere un gran conocimiento de Linux y un mayor tiempo de configuración al ser este proceso más complejo también se ha descartado para este proyecto.

Las otras dos opciones son Pidora, una mezcla de Fedora, desarrollado específicamente para Raspberry Pi y Raspbian, un derivado de Debian Weezy. Ambas, al derivar de sistemas operativos de gran popularidad, han tenido un gran desarrollo de aplicaciones y hay gran cantidad de información sobre el tema.

Por otra parte, estas dos distribuciones pueden ser manejadas mediante línea de comandos o escritorio gráfico. Sin embargo hay dos grandes diferencias, Pidora no permite personalizar las características tanto como Raspbian, lo cual es un punto a tener muy en cuenta si el uso que se le va a dar es el de desarrollo de proyectos, como es el caso. Por otra parte Pidora es mucho más lento de ejecución que Raspbian, lo cual puede afectar seriamente al proyecto, ya que se requiere la mayor velocidad de respuesta posible.

Además Raspbian tiene ventajas añadidas, como que es muy fácil de usar para los que empiezan a utilizar la Raspberry Pi con poca experiencia en Linux, y cuenta con un gran repositorio de software de Debian compatible.

Por estas razones se ha elegido para el desarrollo del proyecto la distribución Raspbian.

3.3 Elección del software. Infraestructura de control. ROS.

Para facilitar la tarea de control del robot se va a utilizar una plataforma de desarrollo o framework. Un framework o infraestructura digital, es una estructura conceptual y tecnológica, que puede servir de base para la organización y desarrollo de software.

Este tipo de software basa su funcionamiento en un sistema operativo externo, al que ceden la gestión de los recursos hardware para centrarse en el control del robot.

Proporcionan una estructura de comunicación por encima del sistema operativo que lo sostiene y también aportan abstracción del hardware, comunicación entre los procesos que se estén ejecutando, acceso a los dispositivos de bajo nivel etc…

Una ventaja de estas plataformas es que incluyen librerías y herramientas que hacen más sencilla la programación, ofreciendo también un acceso más sencillo a los sensores y actuadores.

En el proyecto anterior se basó el desarrollo del control en el proyecto de código abierto Player-Stage, siendo Player un servidor que se encargará de recoger toda la información acerca del robot y controlarlo, mientas que Stage es un simulador de robots en dos dimensiones.

Como ya se ha dicho, la función de Player es obtener toda la información referente al robot, ya sea ésta proveniente de los sensores, actuadores, información adicional como nivel de batería etc. y además permite la interacción con él y su control. Éste se realiza mediante intercambio de mensajes tipo sockets, utilizando un protocolo de comunicación propio. Player incluye librerías que permiten la programación a través de diferentes lenguajes, como por ejemplo C++, Java o Python.

Para hacer simulaciones de los algoritmos desarrollados en Player, en este caso se utiliza Stage, que es un simulador de robots no articulados.

En esta ocasión se ha planteado utilizar otro tipo de plataforma de desarrollo más completa que permita una mayor interactuación con el robot.
La plataforma en la que se ha pensado para ello es ROS. ROS son las siglas de Robot Operating System y al igual que Player, es un framework utilizado para el desarrollo de aplicaciones robóticas. Puede decirse que Player ofrece más controladores de hardware, sin embargo, ROS ofrece más implementaciones de algoritmos, al incluir librerías matemáticas, geométricas, y otras tan útiles como OpenCV para la visión artificial. También soporta diferentes lenguajes como Python, C ++, Java etc.

Adicionalmente, ROS funciona sobre gran cantidad de sistemas operativos, tal como Ubuntu, Fedora, Arch Linux, Windows etc.

Por estos aspectos es bastante similar a las funciones de Player. Sin embargo en comparación con Player, ROS hace más sencillo el control en un entorno distribuido, y el alto nivel está más desarrollado en ROS.

Puede decirse que ROS es más potente y flexible que Player a costa de una mayor complejidad, pero a pesar de ello, ROS está mejor documentado. Debido al gran número de personas que lo utilizan se dispondrá de mucha información disponible en internet, tanto en foros como en la página oficial de ROS http://wiki.ros.org/, donde podrán encontrarse tutoriales para aprender a manejar ROS desde cero.

Entre los beneficios de ROS se encuentra el estar siempre actualizado, ya que cada poco tiempo se saca una nueva versión con nuevas funcionalidades y resolución de los problemas de las versiones anteriores.

Al igual que Player-Stage es un software de libre desarrollo por lo que no supondrá un coste extra para el proyecto.

Por todas estas ventajas se ha decidido desarrollar la navegación del robot dentro de la infraestructura de control Robot Operating System.

Capítulo 4:

Raspberry Pi

En el capítulo anterior se eligió que la plataforma de control de alto nivel sería una Raspberry Pi tipo B, cuyas características se pueden consultar en el anexo B, “Datasheet Raspberry PI”. Este sistema embebido permitirá integrar el control de alto nivel dentro del chasis de la plataforma por lo que se despejará la parte superior de la tapa, que antes ocupaba un ordenador portátil, lugar en el que se podrán transportar otros objetos.

Para integrar todo el control en el robot hay que seguir una serie de pasos, que se explicarán en este capítulo.

4.1 Instalación de Raspbian.

En primer lugar se debe instalar el sistema operativo, más concretamente la distribución Raspbian, basada en Linux. A continuación se explican los pasos que hay que seguir desde un ordenador con sistema operativo Linux.

Se instalará esta distribución a partir de una imagen en la tarjeta SD de 32 GB.

Es importante aclarar que una vez introducida la imagen en la tarjeta se borrará cualquier cosa que contenga, por lo que se debe tener cuidado de que la tarjeta no contenga nada importante. Por esta misma razón hay que tener cuidado a la hora de poner el dominio en el que se quiera sobrescribir el contenido.

Se descargará la imagen de la última versión de la distribución en la página www.raspberrypi.org/downloads donde se pueden encontrar varias imágenes disponibles para descargar, como se puede apreciar en la figura 4.1. De entre ellas se ha elegido la distribución Raspbian que es la elegida para este proyecto.

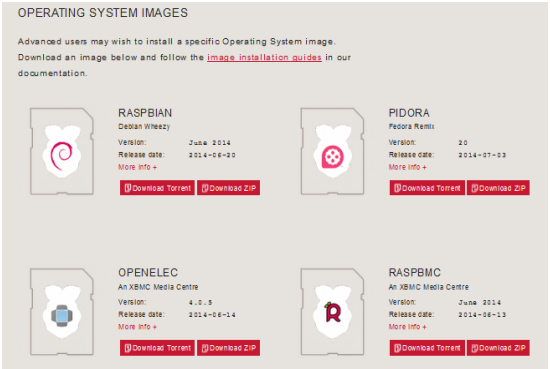


Figura 4.1 Distribuciones para Raspberry Pi.

Se puede comprobar que se ha realizado la descarga correctamente verificando que coincide el código SHA-1 que aparece en la página de descarga, el cual se muestra en la figura 4.2, con el código obtenido al ejecutar el siguiente comando:

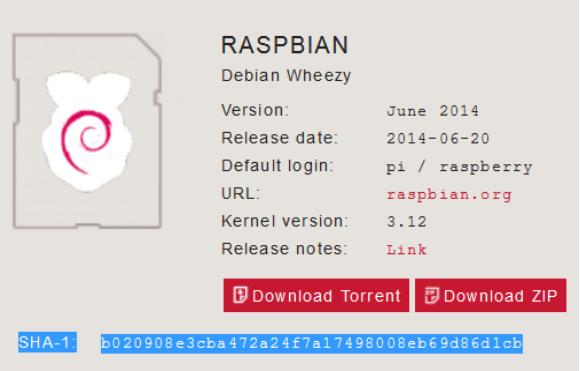


Figura 4.2. Código SHA-1.

```
sha1sum ~/2012-12-16-wheezy-raspbian.zip
```

Suponiendo que ~/ sea la carpeta donde se encuentra la imagen descargada y 2012-12-16-wheezy-raspbian.zip el nombre del archivo, devolverá un código que será igual que el que se muestra en la página si la descarga es correcta.

Una vez descargado el fichero se descomprimirá con el comando:

```
unzip ~/2012-12-16-wheezy-raspbian.zip
```

A continuación hay que ejecutar el comando **df –h**, que mostrará los dispositivos montados en el sistema. Después habrá que conectar la tarjeta SD en el ordenador y volver a ejecutar ese mismo comando. Ahora debería mostrar al menos un nuevo dispositivo. Si aparecen varios es porque la tarjeta tiene particiones.

Antes de seguir se necesitan permisos de administrador, por lo que si no se ha iniciado sesión como root hay que ejecutar los siguientes comandos precedidos del prefijo **sudo**.

El siguiente paso es desmontar la tarjeta. Para ello se utiliza el comando

```
umount /dev/sdd
```

Siendo sdd el nombre de la tarjeta. Si se tiene más de una partición habrá que desmontar todas las particiones:

```
umount /dev/sdd1
umount /dev/sdd2
```

Una vez desmontado se ejecutará el comando que copiará el contenido de la imagen en la tarjeta, sobrescribiendo lo que haya. Un ejemplo de este comando es:

dd bs=4M if=~/2012-12-16-wheezy-raspbian.img of=/dev/sdd

Donde ~/ es la dirección, 2012-12-16-wheezy-raspbian.img el nombre de la imagen y /dev/sdd el nombre de la tarjeta. Es importante escribir el archivo sobre la tarjeta entera y no sobre una de las particiones por lo que aunque se tenga dividida la tarjeta en las particiones sdd1 y sdd2, en el comando solo se pondrá sdd, de manera que se reestructure la división.

El proceso comienza con el comando **dd**. No reporta ninguna información, de manera que el ordenador quedará parado durante un tiempo, hasta que termine de copiarse el archivo. En lugar de utilizar el comando **dd** puede utilizarse el comando **dcfldd** que sí que aportara información.

Una vez terminado el proceso se procederá a ejecutar el siguiente comando **sync** precedido de sudo si es necesario. Esto garantizará que la caché de escritura se vacía y que será seguro extraer la tarjeta del ordenador.

4.2 Configuración de la Raspberry PI.

Una vez instalada la distribución Raspbian en la tarjeta SD solo habrá que insertarla en la Raspberry para empezar a funcionar.

Se puede trabajar de dos maneras, o bien conectando un monitor, teclado y ratón a la Raspberry o bien conectándose de forma remota con un ordenador a través de internet.

En este apartado se va a explicar cómo configurar la Raspberry con el método del monitor, ya que cómo conectarse a internet con la Raspberry se explicará más adelante.

Se conectará la Raspberry con un monitor a través de HDMI o por RCA y se conectará a los puertos USB un teclado y un ratón. Finalmente se conecta la alimentación y arrancará automáticamente. Después de esperar un tiempo pedirá nombre de usuario, que por defecto será “pi” y a continuación la contraseña, que será “raspberry”. Después ya se pueden introducir comandos. Como lo primero que hay que hacer es configurarla habrá que teclear el comando **sudo raspi-config.**

Al teclear ese comando aparecerá el menú de la figura 4.3.

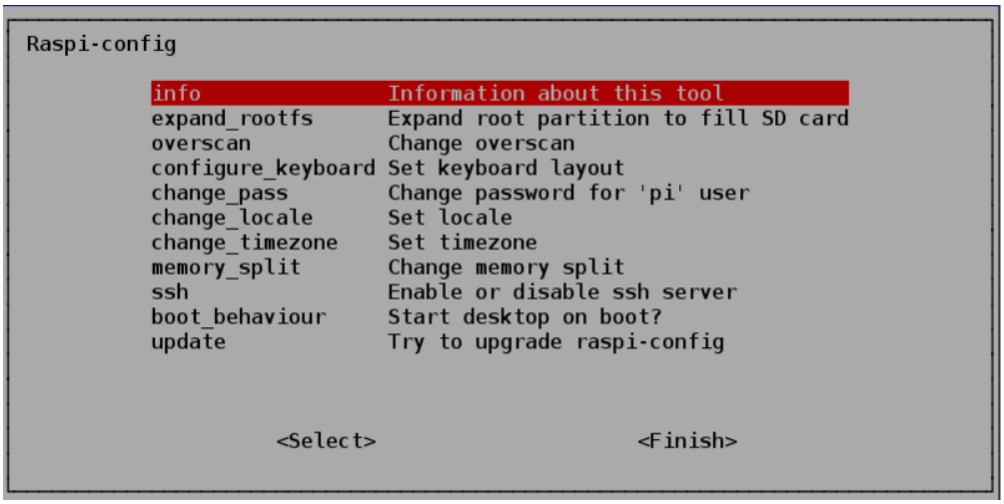


Figura 4.3. Menú de configuración

Ahora se va a explicar para qué sirve cada opción y las opciones que se han tomado en este proyecto.

- **expand_rootfs**: Esto servirá para expandir el tamaño de la partición que se creó al instalar el sistema operativo para aprovechar toda la capacidad de la tarjeta, que en este caso es de 32 GB. En caso de no hacerlo, la capacidad estaría limitada a 2GB.

- **configure_keyboard**: Con esta opción se puede cambiar el teclado por defecto. En este caso se ha especificado que hay un teclado genérico de 105 teclas (Generic 105-key) y a continuación se pulsará en la opción "other" para poder elegir de idioma el español de España (spanish).

- **change_pass**: Para cambiar la contraseña por defecto.

- **change locate**: Se quitará la opción seleccionada con * pulsando la barra espaciadora y se seleccionará es_ES.UTF-8 (español).

- **change_timezone**: Para elegir la zona horaria. Se seleccionará Europa y dentro de ella, Madrid.

- **memory_split**: Con esta opción se puede cambiar la asignación de memoria que se dedica a los graficos y la que tiene otros usos. En este proyecto no se ha modificado esta opción.

- **ssh**: En este apartado se pondrá Enable, para habilitar la conexión ssh.

- **boot_behaviour**: Este apartado da la opción de iniciar el escritorio gráfico cada vez que se conecte la rapsberry. En este caso se ha habilitado, pero en caso de no hacerlo, en cualquier momento se puede teclear el comando **startx** y se iniciará el escritorio gráfico.

- **update**: Se actualizará a la última versión.

Una vez terminado se pulsará Finish y la Raspberry se reiniciará. En caso de que no se reinicie automáticamente habrá que reiniciarla manualmente con el comando **sudo reboot**.

Con esto finaliza el apartado de configuración. Se podrá cambiar la configuración en cualquier momento que se quiera, tecleando de nuevo en un terminal el comando **sudo raspi-config**.

4.3 Conexión a internet. IP fija y WIFI.

El objetivo de este apartado es explicar cómo conectar la Raspberry a internet, tanto mediante Ethernet como con WIFI.

Se va a trabajar con Ethernet mientras se esté desarrollando la aplicación ya que es mucho más rápido que internet mediante WIFI. Sin embargo, una vez la aplicación esté completa funcionará mediante WIFI, ya que será necesario que haya comunicación entre la Raspberry y el ordenador, y la Raspberry Pi irá integrada dentro de la plataforma. Como no se puede depender de que la capacidad de desplazamiento del robot se limite a la longitud de un cable, la comunicación se realizará mediante WIFI, que proporcionará la velocidad suficiente para el buen funcionamiento del sistema.

En primer lugar se va a explicar la conexión mediante Ethernet, ya que es la primera que se va a utilizar.

Se usará una conexión mediante IP fija, debido a que este es el tipo de conexión que se utiliza en el centro tecnológico Cartif, donde se ha desarrollado este proyecto. Tener una IP fija también aporta la ventaja de que la Raspberry Pi siempre se conectará a la IP que se le fije, por lo que en un entorno educacional, donde presumiblemente habrá más de una Raspberry Pi conectada se podrá encontrar el dispositivo deseado sin problemas, si en vez de trabajar con monitores se hace a través de conexión SSH.

Para configurar el tipo de conexión se deben modificar archivos de la Raspberry. lo primero que se debe hacer es abrir el administrador de archivos con permisos de administrador. Esto se consigue abriendo una terminal e introduciendo el comando **sudo pcmanfm**.
Una vez abierto el administrador de archivos se abrirá la carpeta etc, que cuelga directamente del directorio raíz, y dentro de ella se busca la carpeta network, que contiene un archivo de texto llamado interfaces. Llegado a este punto se puede hacer una copia de seguridad del archivo original con otro nombre y a continuación modificar el archivo con el nombre "interfaces". De esta manera se conservará la información original, que sirve para conectarse a una red con IP variable.

Una vez realizada la copia de seguridad se cambiará el archivo "interfaces" para que siga esta estructura.

```
auto lo
iface lo inet loopback
iface eth0 inet static

address 192.168.107.90
netmask 255.255.255.0
network 192.168.107.0
broadcast 192.168.107.255
gateway 192.168.107.59
```

Con las primeras líneas se conseguirá conectarse automáticamente a internet y se especifica que debe ser con una IP fija (static). Las siguientes líneas son datos de la red a la que se desea que se conecte, siendo address la dirección IP a la que se desea que se conecte. El caso anterior es un ejemplo.

Por último habrá que reiniciar la Raspberry Pi. Cuando vuelva a encenderse se conectará automáticamente a la red con la IP especificada.

A continuación se va a explicar la conexión a internet mediante WIFI. El proceso es similar al que ya se ha explicado para configurar la conexión mediante cable de red, sin embargo en esta ocasión es necesario un pincho WIFI que haga de antena y permita acceder a las redes.

Para este proyecto se ha buscado una antena WIFI con conexión USB perfectamente compatible con Raspberry PI, ya que no todas lo son. La elegida ha sido D-link DWA-121 N150 chipset Realtek RTL8188CUS, ya que además de ser compatible es muy compacta como se puede apreciar en la figura 4.4.

Figura 4.4. Antena Wifi- USB

Una vez elegida la antena hay que configurarla.

Al igual que en el caso anterior hay que abrir el administrador de archivos como administrador y editar el archivo "interfaces". Para conservar la configuración de la conexión mediante Ethernet lo que se ha hecho ha sido cambiar el nombre del archivo a "interfaces_eth" y crear un nuevo archivo llamado "interfaces" para conservar así ambas configuraciones y poder cambiar con facilidad de una a otra.

En el nuevo archivo se introducirán las siguientes líneas.

```
auto lo
iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

Con ellas se podrá configurar la red a través de un programa específico ya instalado en la Raspberry, que habrá que ejecutar después de reiniciarla.

Figura 4.5. Configuración Wifi

Para poder configurar la WIFI se insertará la antena WIFI, y no se conectará el cable de Ethernet ya que en ese caso no funcionará adecuadamente y no se podrá configurar. Por esta razón sólo se podrá realizar este apartado si se conecta el ordenador a un monitor y no mediante conexión SSH.

Una vez iniciada la Raspberry se ejecuta el programa Wi-Fi config que se encuentra en el escritorio y aparecerá el cuadro mostrado en la figura 4.5.

El siguiente paso es indicarle al programa que busque las redes WIFI disponibles mediante la opción "Scan", con lo que aparecerá el cuadro mostrado en la figura 4.6 con una lista de redes disponibles.

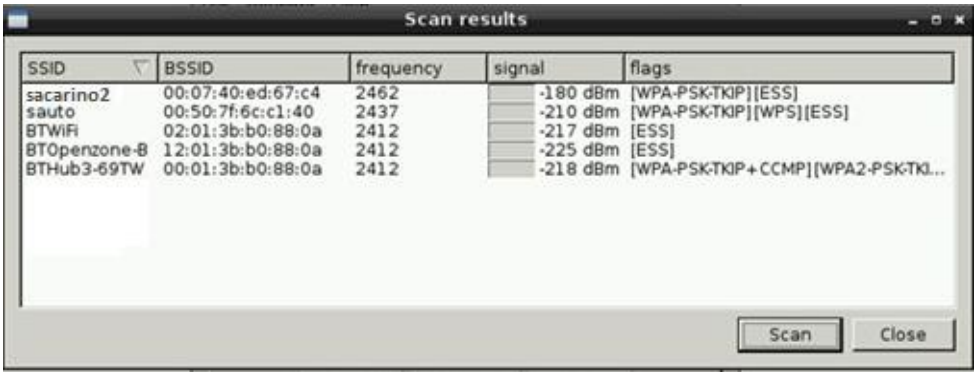


Figura 4.6. Redes WIFI disponibles

Una vez haya escaneado aparecerán las redes al alcance, entre las cuales habrá que escoger una haciendo doble clic sobre la seleccionada.
Se abrirá un cuadro como el de la figura 4.7, en el que habrá que introducir la contraseña en el apartado PSK.

Hecho esto se pulsará el botón "Add" para terminar.

Ahora ya debería estar conectado. Para comprobarlo se ejecutará en una terminal el comando **ifconfig**. Debería aparecer una dirección IP que en este caso ha sido 192.168.2.136. Esta IP es a la que se conectará siempre la Raspberry en caso de estar disponible. Ya que la IP no tiene por qué cambiar se puede apuntar para conectarse remotamente a través de WIFI.

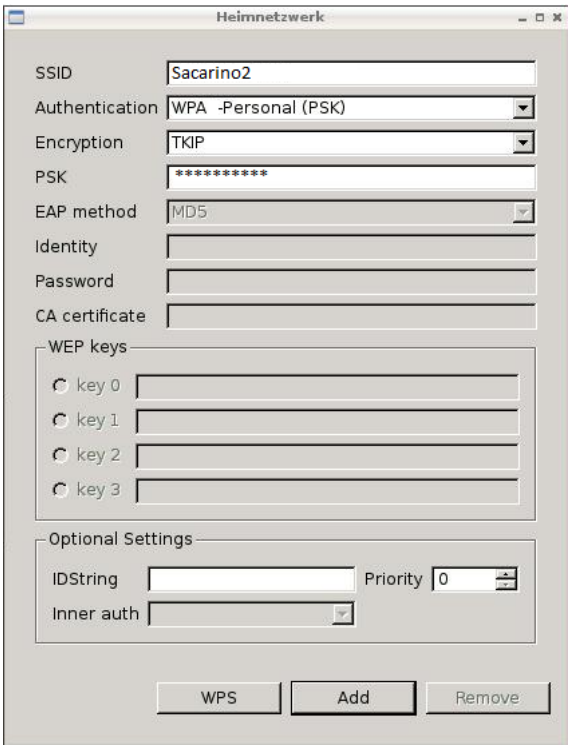


Figura 4.7. Conexión a una Red Wifi

Antes de reiniciar o apagar la Raspberry hay que salvar la configuración. De esta manera la Raspberry se conectara automáticamente si tiene insertado el USB y la red que se acaba de añadir está disponible.

4.4 Conexión remota.

Hay dos maneras de conectarse a la Raspberry Pi de forma remota a través de un ordenador, mediante SSH o mediante escritorio remoto.

La primera opción y la que se va a explicar en primer lugar es la conexión SSH.

Para poder conectarse con la Raspberry mediante SSH se debe haber habilitado la opción SSH en la configuración inicial de la Raspberry, o bien cambiar ésta accediendo al menú de configuración mediante el comando "sudo raspi-config" a través de un monitor.

Otra condición importante es que la Raspberry debe estar conectada a la misma red que el ordenador con el que se va a acceder. Es decir, si la conexión va a ser mediante Ethernet los dos dispositivos deben estar conectados a internet a través de la misma red de Ethernet o si se va a utilizar WIFI ambos deben estar conectados a la misma red WIFI.

Ya debería ser conocida la IP a la que se ha conectado la Raspberry Pi, pero si se desconoce o se quiere comprobar se deberá ejecutar en un terminal de la Raspberry el comando **ifconfig,** que mostrará la IP con la que se ha conectado el dispositivo.

Sabiendo ésta dirección se ejecutará en un terminal (tanto en Ubuntu como en virtual box a través de Windows) la instrucción " **ssh -X pi@192.168.107.23**" (Cambiar esta IP por la de la la Raspberry en la red), a continuación edirá la contraseña de la Raspberry Pi.

Una vez se introduzca ya se habrá establecido la conexión a través de SSH.

Ahora se expondrá el segundo tipo de conexión, la conexión mediante escritorio remoto. Para poder realizar este tipo de conexión hay que instalar un programa en la Raspberry. Esto se va a hacer mediante comandos, que podrán ser ejecutados bien en la Raspberry directamente o bien conectándose a ella mediante SSH.

En primer lugar se actualizarán los repositorios de software mediante el comando:

```
sudo apt-get update
```

Y a continuación se instalará el servidor del repositorio que se va a usar:

```
sudo apt-get install tightvncserver
```

El siguiente paso es ejecutar el servidor, que demandará una contraseña. Esta contraseña será la usada al establecer la conexión de manera remota. Después se tecleará lo siguiente:

```
/usr/bin/tightvncserver
```

Existe también la opción de crear una contraseña que solo permita la lectura de los documentos que contiene la Raspberry. En este caso no se ha utilizado.

Una vez que se inicie el servidor, éste mostrará qué escritorio virtual ha sido iniciado.

Por ejemplo mostrará:

```
New 'X' desktop is raspberrypi:1
```

Con esto se está indicando que el nuevo escritorio es el 1 por lo que en este caso habrá que conectarse al servidor usando la IP de la Raspberry PI seguida de ":1"-

Ahora se quiere conseguir que este programa se ejecute automáticamente al iniciarse la Raspberry Pi, de forma que se pueda conectar siempre mediante el escritorio virtual.

Para conseguirlo es necesario un fichero de inicio, que se puede descargar usando los siguientes comandos:

```
wget http://www.penguintutor.com/otherfiles/tightvncserver-init.txt
```

```
sudo mv tightvncserver-init.txt /etc/init.d/tightvncserver
```

En caso de haber cambiado el nombre de usuario se deberá modificar el archivo que se acaba de descargar y en la línea 16, donde pone "export

USER=’pi’” cambiar pi por el nombre de usuario que se haya puesto (excepto root). En este caso no se ha modificado el nombre de usuario por lo que no será necesario ningún cambio.

El siguiente paso es hacer ejecutable el fichero que se ha descargado introduciendo el siguiente comando:

```
sudo chmod 755 /etc/init.d/tightvncserver
```

Por último se añadirá el script a los niveles de ejecución mediante la instrucción:

```
sudo update-rc.d tightvncserver defaults
```

Siguiendo estos pasos, TightVNC se iniciará al arrancar la Raspberry. Se puede comprobar de dos maneras. O bien reiniciando la Raspberry Pi o bien deteniendo tightvnc y volviendo a iniciarlo de la siguiente manera:

```
sudo /etc/init.d/tightvncserver stop
sudo /etc/init.d/tightvncserver start
```

Una vez comprobado que funciona se abrirá en ubuntu el cliente de escritorio remoto Remmina, en el cual habrá que crear una nueva conexión.

Como se puede apreciar en la figura 4.8, en dicha conexión hay que elegir como protocolo el RDP (Remote Desktop Protocol) e introducir los datos de la Raspberry Pi que se está utilizando, tal como la direccion IP, el usuario y la contraseña.

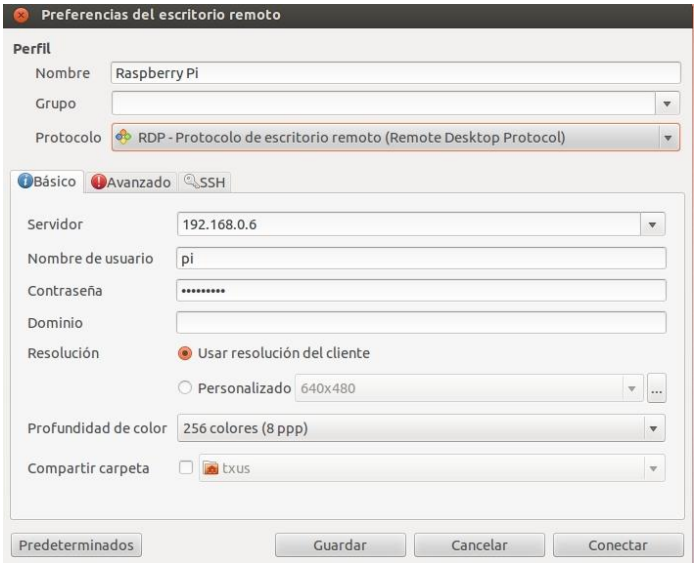


Figura 4.8. Configuración de escritorio remoto

Para terminar se ejecutará la nueva conexión que se acaba de configurar y, si se han seguido todos los pasos correctamente, aparecerá el escritorio de la Raspberry Pi en la pantalla del ordenador, como se puede apreciar en la figura 4.9.

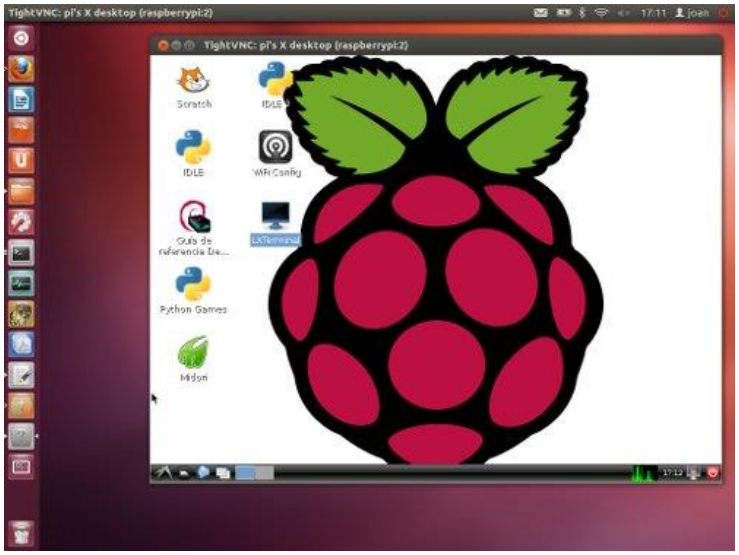


Figura 4.9. Escritorio de la Raspberry Pi

De esta forma finaliza el apartado de conexión remota.

4.5 Nombre de red.

Como este robot tiene fines educativos es de suponer que puede haber varias plataformas funcionando al mismo tiempo. Es por ello que se ha investigado un método para poder reconocer fácilmente cual es cada Raspberry conectada a la red ya que la IP no es fiable al no ser fija.
Para solucionar éste problema se podrá activar el nombre de red. Para ello hay que instalar un servicio llamado Bonjour. Se realizará con los siguientes comandos:

```
sudo apt-get update
sudo apt-get install libnss-mdns
```

Al instalar este programa no será necesario conocer la IP de la Rapsberry, solo su nombre de red, que por defecto será “raspberrypi.local”.

De este modo, para acceder a la Raspberry mediante SSH a través de Linux se tecleará:

```
ssh pi@raspberrypi.local
```

Sin embargo aparece un segundo problema. Todas las Raspberry Pi tendrán el mismo nombre. Esto afectaría mucho en caso de tener varios dispositivos en una misma red, ya que sólo la primera que consiga conectarse tendrá acceso a internet y se quedará con el nombre por defecto, mientras que el resto no conseguirá conectarse.

Para solucionar este inconveniente se va a explicar cómo cambiar el nombre de host (o hostname) del dispositivo. El hostname es el nombre que adquiere una máquina dentro de una red local. Para cambiarlo habrá que seguir los pasos que se enumeran a continuación.

Lo que se debe hacer en primer lugar es modificar el archivo host, que se encuentra dentro de la carpeta etc. Se puede hacer abriendo el administrador de archivos o bien ejecutando la siguiente línea en un terminal:

```
sudo nano /etc/hosts
```

Lo que se hará será modificar la línea en la que pone "*127.0.1.1 raspberrypi*", cambiando el nombre por defecto "raspberrypi" por el nombre de host que se quiera adoptar.

Por último habrá que cambiar el archivo hostsname, también en la carpeta etc y sustituir la palabra "raspberrypi" por el nombre de hostname que se haya puesto en el otro archivo.
De esta manera se podrá tener un nombre diferente para cada dispositivo conectado.

Capítulo 5:

Robot Operating System

En este capítulo se darán unas nociones básicas de ROS, imprescindibles para comprender los detalles de la navegación del robot y poder ejecutarla en la Raspberry Pi partiendo del software desarrollado. Para la elaboración de esta capítulo se ha tenido muy en cuenta los tutoriales oficiales.

5.1 Instalación de ROS Groovy en Raspbian.

En primer lugar indicará como instalar ROS Groovy, que es la versión que se va a utilizar, en Raspbian, ya que ese fue el sistema operativo para la Raspberry que se eligió.

La versión Groovy será la utilizada en este caso, debido a que está desarrollado para la distribución Raspbian de Raspberry Pi, al contrario que la última versión, Hydro.

Para comenzar con la instalación debe acudirse al apartado correspondiente de la página oficial de ROS, que es el siguiente: http://wiki.ros.org/groovy/Installation.

Una vez en esta página aparecerá una lista de distribuciones en las que puede funcionar, (figura 5.1), de la cual se elegirá Raspbian.

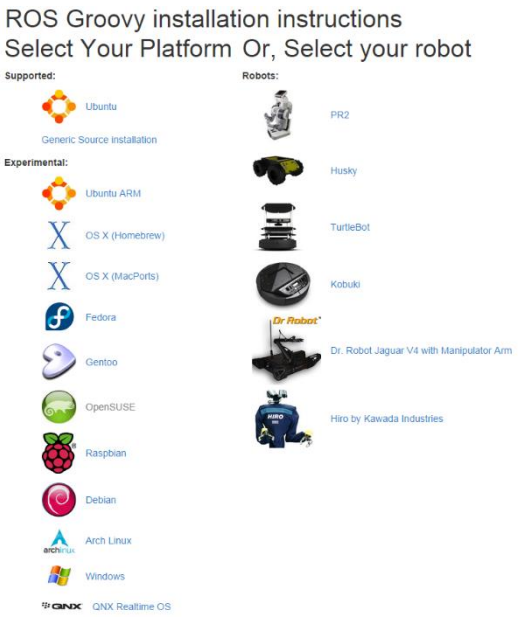


Figura 5.1. Listado de sistemas que trabajan con Groovy.

El repositorio que se va a instalar es experimental, y contiene la mayor parte de la distribución. Con este repositorio será suficiente para los requerimientos de la navegación que se ejecutarán en la Raspberry Pi.

Para instalar este repositorio deben seguirse los pasos que se describen a continuación:

Abrir una terminal en la Raspberry Pi para ejecutar los siguientes comandos.

Añadir el repositorio a las fuentes de apt.

```
$ sudo sh -c 'echo "deb http://64.91.227.57/repos/rospbian wheezy main" > /etc/apt/sources.list.d/rospbian.list'
```

A continuación proporcionar la clave:

```
$ wget http://64.91.227.57/repos/rospbian.key -O - | sudo apt-key add
```

Recargar las fuentes de apt para que se tengan en cuenta las recién añadidas.

```
$ sudo apt-get update
```

A continuación se instalarán los paquetes de ROS de este repositorio.

```
$ sudo apt-get install ros-groovy-ros-comm
```

Por último en el archivo de configuración inicial llamado .bashrc de Raspberry, se añade la dirección de los paquetes de ROS, de manera que no habrá que volver a realizar este paso cada vez que se inicie una nueva sesión.

```
$ source /opt/ros/groovy/setup.bash
```

Para comprobar que se ha instalado correctamente se intentará realizar la siguiente acción

```
$ roscore
```

Si se ejecuta correctamente se habrá instalado de forma adecuada el repositorio Comm, de Groovy.

5.2 Nociones básicas de ROS.

Una vez instalado, se procederá a impartir unas nociones de ROS que permitan comprender las explicaciones que se hagan más tarde, en este mismo proyecto, sobre la navegación y el modo de ejecutarla.

5.2.1 Espacio de trabajo, paquetes y sistema de compilación.

Un espacio de trabajo o workspace es una carpeta donde se alojarán todos los módulos que se desarrollen. En este caso se creará un espacio de trabajo de catkin, que es el sistema de compilación que se va a utilizar. Con este sistema, se compilarán a la vez todos los paquetes que estén contenidos en el espacio de trabajo.

Para crear el espacio de trabajo deberá crearse primero una carpeta (en este caso se la ha llamado catkin_ws) y dentro de ella otra llamada src. A continuación se iniciará esa carpeta como espacio de trabajo:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
```

Dentro de la carpeta src debe guardarse paquetes que se van a utilizar, y que se nombrarán más adelante en el capítulo 8.

Los paquetes son la unidad de organización de software que maneja ROS. Cada paquete puede contener bibliotecas, ejecutables, scripts u otros artefactos, Además necesita incluir un documento llamado package.xml y otro llamado CMakeLists.txt para ser considerado un paquete.

Estos documentos hacen una descripción del paquete. Sirven para definir las dependencias entre paquetes e incluyen información como la versión, mantenedor, licencia, etc

El paquete más simple posible podría tener este aspecto:

```
my_package /
  CMakeLists.txt
  package.xml
```

Por lo que un espacio de trabajo podría tener este aspecto:

```
catkin_ws/
  src/
    CMakeLists.txt
    paquete_1/
      CMakeLists.txt     -- CMakeLists.txt del  paquete _1
      package.xml        -- Package del paquete _1
    ...
    paquete_n/
      CMakeLists.txt     -- CMakeLists.txt file for paquete _n
      package.xml        -- Package manifest for paquete _n
```

Una vez guardados todos los paquetes que se van a utilizar en este proyecto hay que compilarlos, y para ello hay que desplazarse hasta el workspace y ejecutar el comando **catkin_make**:

```
$ cd ~/catkin_ws/
$ catkin_make
```

Esta instrucción creará dos carpetas dentro de catkin_ws, devel y build. En la carpeta devel, se pueden encontrar varios archivos de configuración. Se seleccionará uno de ellos para hacer funcionar los paquetes contenidos en el entorno de trabajo.

```
$ source devel/setup.bash
```

5.2.2 Funcionamiento general. Definición de nodos, topics, mensajes y servicios. Uso de rostopic. Herramienta rqt.

Los paquetes de ROS incluyen en su interior Nodos, Topics y Servicios. A continuación se describirá cada uno de los elementos, para tener un mayor conocimiento del funcionamiento de ROS.

Los nodos son la unidad de procesamiento, son los ejecutables contenidos en los paquetes. Utilizan librerías de ROS para comunicarse entre ellos. Pueden mandar mensajes mediante Topics y proporcionar o utilizar servicios.

Topics son el canal de comunicación entre los nodos. Un nodo publicará mensajes en un topic y otros nodos se suscribirán a ese topic si quieren recibir los mensajes.

Los mensajes son estructuras de datos que se intercambian entre los nodos. Son el contenido de los topics.

Los servicios permiten que haya una arquitectura cliente-servidor entre los nodos. El nodo servidor se mantiene a la espera de solicitudes, las procesa y responde al nodo cliente. En este proyecto no van a utilizarse.

Para obtener más información acerca de los topics y mensajes se puede utilizar el comando **rostopic**. Esta instrucción muestra información acerca de los topics. Cuenta con diferentes opciones como:

- **rostopic echo**: muestra los datos publicados sobre un tema. Se debe ejecutar este comando por ejemplo si se quiere visualizar los datos provenientes de un topic.

- **rostopic list**: devuelve una lista de todos los Topics que se están utilizando en ese momento.

- **rostopic pub**: publica datos en el topic que se especifica. Por ejemplo

rostopic pub [topic] [tipo_de_mensaje] [argumentos]

Para mostrar con mayor claridad el funcionamiento del conjunto se utilizará la herramienta de ROS rqt, en concreto rqt_graf, que es parte del paquete rqt.

Rqt_graph que muestra los nodos y los temas actualmente en ejecución, creando un gráfico de lo que está pasando en el sistema.

Por ejemplo, en la figura 5.2 pueden verse dos nodos ejecutándose, turtlesim_node y turtle_teleop_key, comunicados mediante el topic / turtle1 / command_velocity.

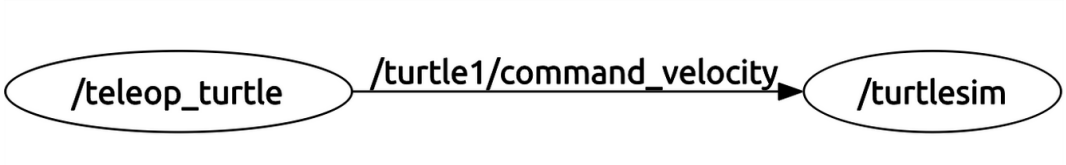


Figura 5.2. Ejemplo ejecución rqt_graph.

El nodo teleop_turtle es el que emite los mensajes mientras que turtlesim está suscrito al topic y los recibe.

5.2.3 Lanzamiento de nodos. Uso de roslaunch.

Se continúa con el método de lanzamiento de los nodos.

Antes de lanzarse cualquier nodo debe ejecutarse en una terminal "**roscore**". Roscore es un conjunto de nodos y programas que deben lanzarse antes que cualquier otro elemento. Los programas que lanza son: el nodo maestro, que se encarga de que los nodos se encuentren los unos a los otros, el Parameter Server y el nodo rosout.

Una vez se esté ejecutando roscore se puede lanzar cualquier otro nodo, cuyas dependencias lo permitan, mediante el comando **rosrun**.

Rosrun permite usar el nombre del paquete para ejecutar directamente un nodo, sin necesidad de conocer la ruta del paquete. Un ejemplo de ejecución sería:

```
$ rosrun eusebio eusebio
```

Por otra parte si se desea iniciar varios nodos al mismo tiempo puede crearse un archivo de lanzamiento, que tiene la extensión .launch.

Lo normal es que estor ficheros estén ubicados en una carpeta llamada "launch", en el interior uno de los paquetes que se va a ejecutar cuando se lance.

Un ejemplo de ficheros de este tipo es el siguiente:

```
<launch>
         <node pkg="gpmrc_node" type="gpmrc_node" name="gpmrc_node" cwd="node"
       args="-port /dev/ttyUSB0 -baud 38400" respawn="true" > </node>

         <node pkg="eusebio" type="eusebio" name="eusebio" cwd="node" args="-f
       $(find eusebio)/conf/robot_parameters.yaml -ch_pose $(find
       eusebio)/conf/ChargingPose.yaml -status $(find
       eusebio)/conf/RobotStatus.yaml" respawn="true"> </node>
</launch>
```

En este fichero, llamado "eusebio.launch" y guardado dentro del paquete "eusebio" se lanzan los nodos gpmrc_node y eusebio, y se ejecuta de la siguiente manera:

```
$ roslaunch eusebio eusebio.launch
```

Con esto se puede dar por concluidas estas lecciones básicas.

Capítulo 6:

Sistema de navegación

6.1 Descripción general del sistema de navegación.

En este capítulo se va a describir de forma detallada el sistema de navegación del robot que ha sido desarrollado, de manera que al final del capítulo se hayan comprendido todos los pasos seguidos para el funcionamiento del movimiento autónomo.

En primer lugar se hará una descripción más general, para dar una visión global del funcionamiento, y más adelante se expondrán las tareas de las que se encarga cada nodo. La definición de nodo y de otros términos de ROS están descritos en el capítulo anterior, por tanto es conveniente leerlo para tener unas nociones básicas del sistema antes de comenzar con este capítulo ya que el sistema de navegación está desarrollado con la infraestructura de control Robot Operating System.

Para el desarrollo de la navegación autónoma se han utilizado varios nodos, que se comunicarán entre ellos mediante topics. De esta forma, cada uno realizará una función distinta y le mandarán un mensaje con la información necesaria al siguiente nodo a través de los topics que los conectan.

Por esta razón, hay nodos que no pueden funcionar de manera autónoma, sino que necesitan que otro se esté ejecutando para que su acción tenga repercusión en el robot. Es el caso del nodo Eusebio, que recibe una dirección que debe seguirse, la interpreta y la transforma en una velocidad para cada uno de los motores. Sin embargo no funcionará a menos que se esté ejecutando también el nodo GPMRC, que es el que transforma las velocidades y direcciones de los motores en el código ASCII que puede interpretar el hardware.

El nodo Hokuyo, sin embargo, puede funcionar de manera autónoma ya que solamente se encarga de leer la información que produce el sistema laser.

Se va a comenzar con la descripción del sistema de navegación desde el más alto nivel y se irá descendiendo hasta llegar al nivel de hardware.

En primer lugar lo que se debe hacer es crear un mapa del entorno por el cual se va a desplazar de forma autónoma. Esto se hace a través del nodo gmapping, que ya viene incluido en ROS y por ello no ha habido que desarrollarlo. Lo que se hará será lanzar éste nodo y el nodo Hokuyo. El nodo Hokuyo se encargará de leer la información que proporciona el sistema laser. Mientras, el nodo gmapping se encargará de interpretar esa información, junto con los datos de odometría que proporciona el robot y la irá dibujando en una imagen. A continuación lo que hay que hacer es desplazar el robot lentamente de forma manual, parando cada poco tiempo, de manera que el nodo pueda reconocer los obstáculos que más tarde se deben evitar.

Una vez reconocido el terreno en el que tendrá que moverse Eusebio se guardará el mapa para poder usarlo más tarde. Esto se hará mediante el nodo map_server, que está incluido en el stack navigation. Este nodo permite guardar los mapas que crea gmapping en un archivo, y volverlos a cargar cuando es necesario.

A continuación ya se puede navegar. Para ello se debe cargar con el nodo map_server el mapa del entorno que previamente se ha creado y un fichero con la trayectoria que se quiere seguir en ese entorno. El nodo AMCL ubicará el robot dentro de ese mapa gracias a la colocación de los obstáculos de alrededor. Por último el nodo path_follower ubicará los puntos de la trayectoria en el mapa y calculará la ruta a seguir para pasar por esos puntos esquivando los objetos que aparecen en el mapa. Una vez calculada la ruta, le indicará al nodo Eusebio la dirección a seguir, emulando las señales de salida del joystick para el control manual mandándole un mensaje en el que aparecerán los datos sobre el estado de los sticks y de los botones.

El nodo Eusebio se encarga de transformar el mensaje tipo joystick en dos velocidades, la que debe seguir cada motor para avanzar en la dirección que se ha indicado en el mensaje. Una vez transformadas las órdenes enviará las velocidades de los motores al nodo Gpmrc_node. Además se encarga de leer las señales que proporcionan los sonars y bumpers, y si algún objeto está excesivamente cerca o algún bumper está pulsado detendrá el robot.

El nodo Gpmrc_node recibe del nodo Eusebio una velocidad para cada motor. La tarea de este nodo es transformar esas velocidades en datos que puedan ser interpretados por el hardware.

Por último está el nodo Serial_port, sin el cual la comunicación entre el hardware y el control de alto nivel no sería posible, de manera que las órdenes que envía el nodo Gpmrc_node deben pasar primero por el nodo Serial_port para poder ser ejecutadas correctamente.

En la figura 6.1 se muestra un esquema de todos los nodos que intervienen con las relaciones entre ellos.

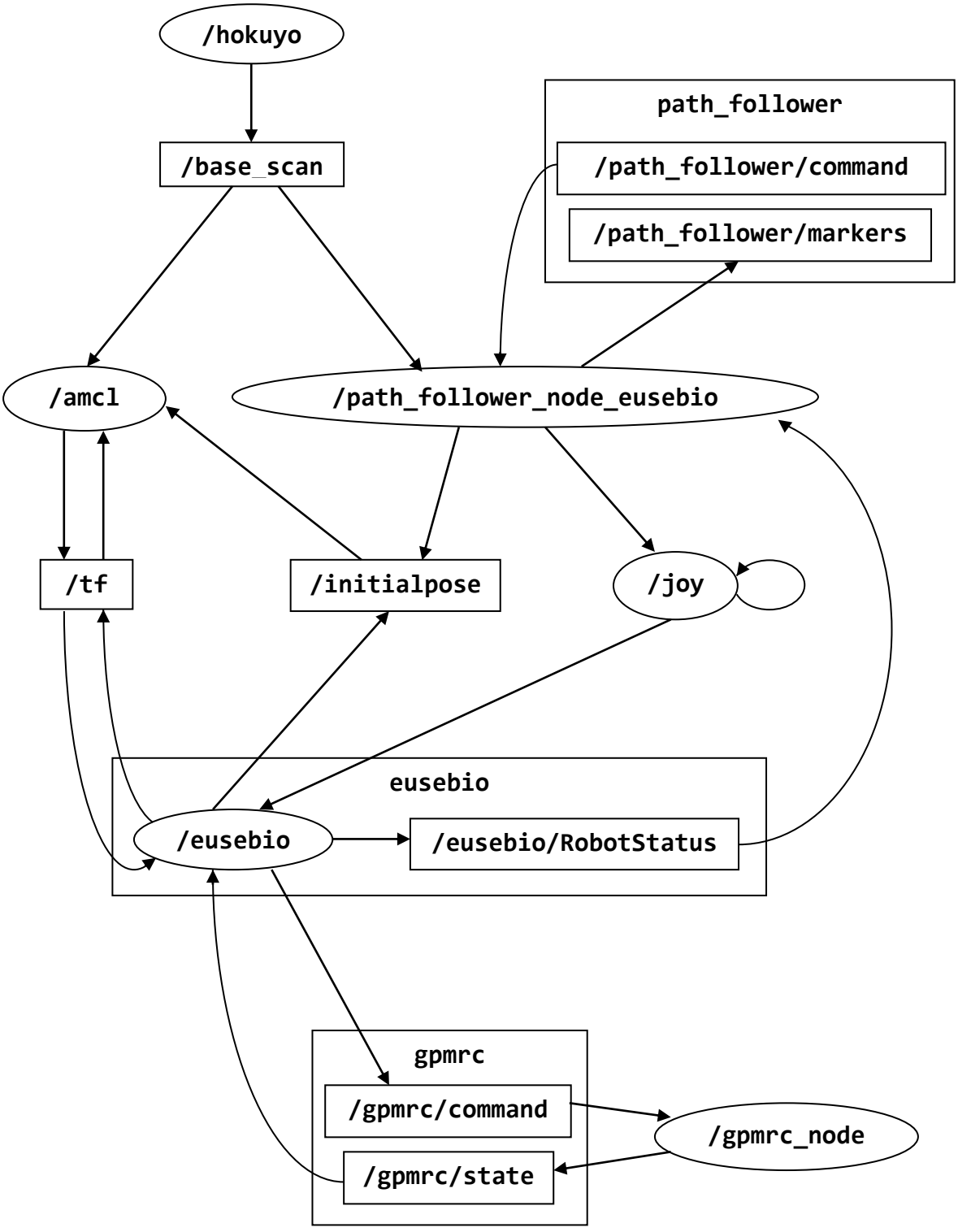


Figura 6.1. Esquema general de los nodos

6.2 Hokuyo.

Este nodo permitirá la comunicación entre la Raspberry Pi y el sistema laser. Es el encargado de leer la información que proporciona el láser. Esta información se compone de una sucesión de distancias a las que se detectan objetos. Se pueden visualizar estos datos a través de la terminal si mientras se está ejecutando el nodo se usa el comando:

rostopic echo /base_scan

Una vez hecho aparecerá una sucesión de los datos en pantalla, de la forma mostrada en la figura 6.2.

Figura 6.2. Lectura del laser sin formato

Como se puede ver, es completamente ilegible a simple vista, por lo que es necesario algo que interprete esta información para que sea comprensible para las personas y para el robot. La herramienta rviz de ROS representa de forma gráfica estos datos de distancias.

6.3 Gmapping.

El nodo gmapping está incluido en el stack SLAM_gmapping, que no ha sido desarrollado en este proyecto sino que ya estaba incluido en el repositorio de ROS.

Este nodo proporciona SLAM (localización y mapeo simultáneo), basada en la información proporcionada por el láser. De esta manera este nodo interpreta la información que llega del sistema laser en forma de distancias y representa los datos de forma gráfica.

En la figura 6.3 puede verse la representación de los datos que está proporcionando el sensor en una imagen. La imagen puede ser visualizada a través de la herramienta de ROS rviz.

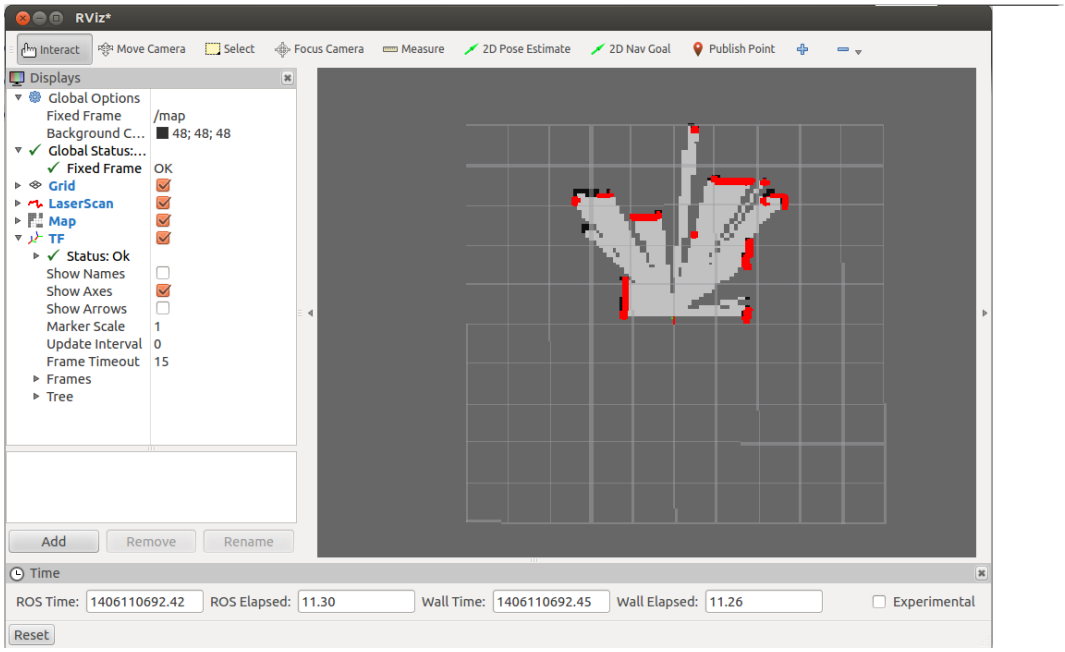


Figura 6.3. Creación del mapa en el ordenador

A continuación se muestra en la figura 6.4 el espacio real en el que se encuentra el robot en el momento de la toma de datos que se han representado en la figura anterior.

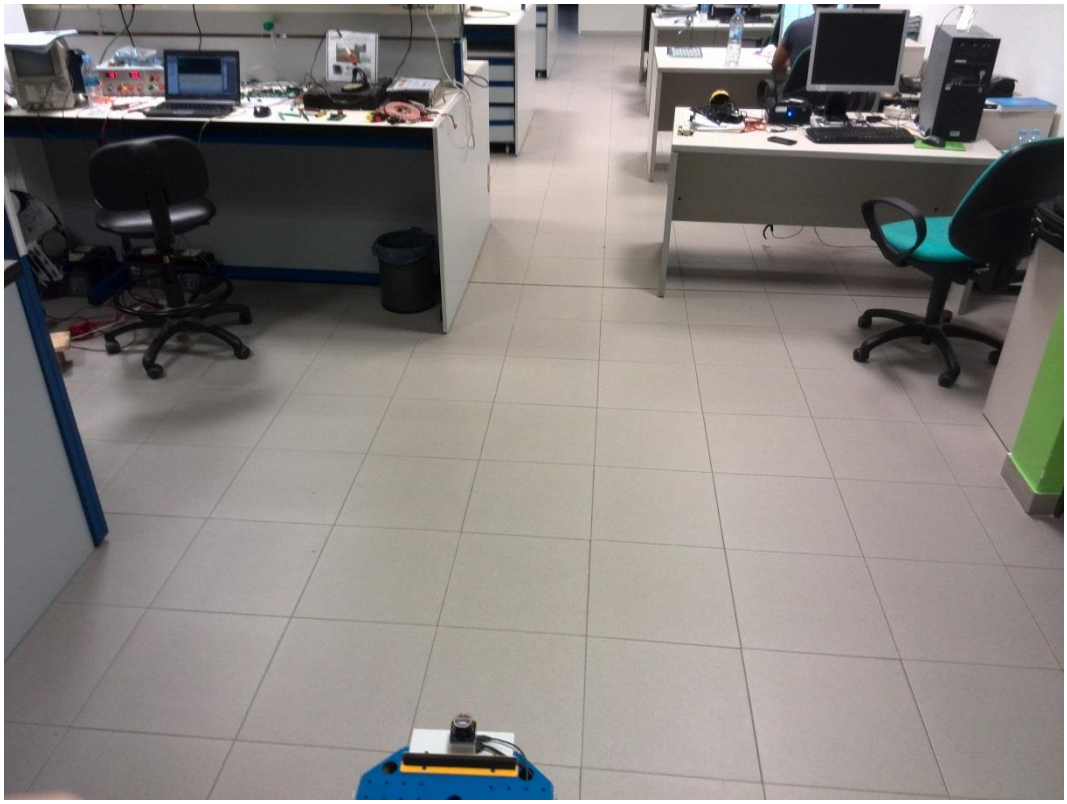


Figura 6.4. Escenario real

Como se puede ver se corresponde de forma muy fiable el espacio físico con los resultados de la lectura del sensor, detectando ambas mesas, las paredes laterales y un espacio vacío en medio, que es el pasillo.

Es por la correspondencia encontrada que se podrán crear mapas del entorno por el que se quiera navegar en el los que aparezcan situados de forma muy fiable los obstáculos físicos.

Con este nodo se creará el mapa del espacio en el que se quiere que el robot se mueva autónomamente guardando los datos obtenidos mediante una navegación manual previa. Se moverá el robot a una velocidad lenta mediante el joystick, parando cada poco tiempo para que el robot reconozca los obstáculos y los ubique en el mapa. Cuando el resultado sea satisfactorio se guardará el resultado de la lectura en un archivo que podrá ser cargado más adelante, antes de empezar a navegar. La creación de un mapa se explica más detalladamente en el siguiente capítulo, donde se encuentra un manual de usuario, en el apartado “Creación de un mapa de navegación”.

El resultado de la ejecución de este nodo y la navegación manual en el departamento de robótica y visión artificial de Cartif es el mostrado en la figura 6.5.

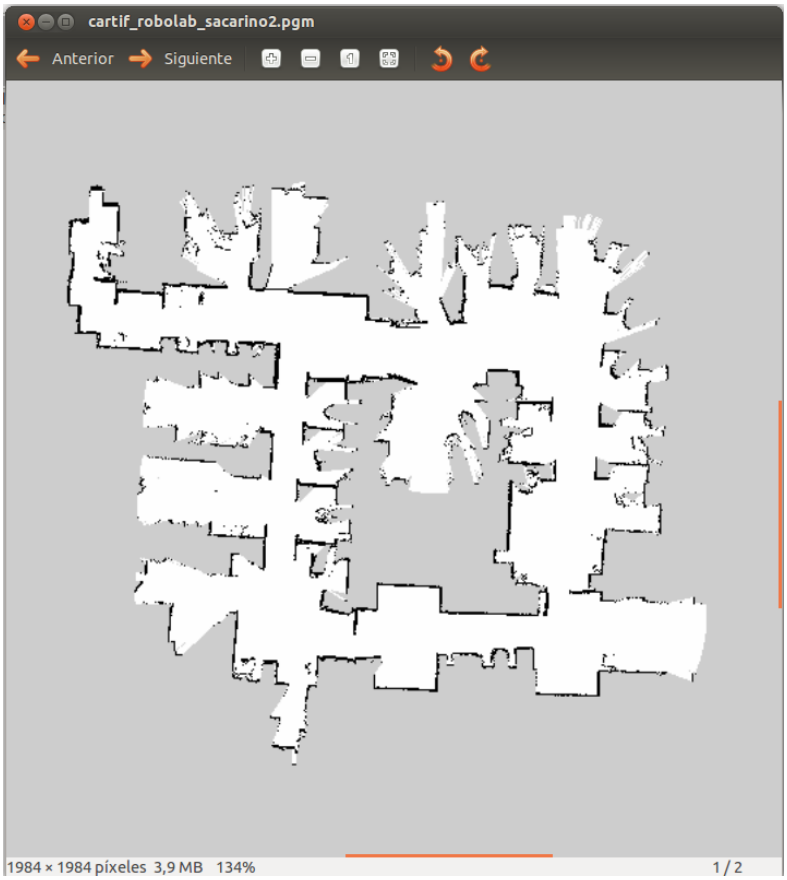


Figura 6.5. Mapa creado en el centro de trabajo

6.4 Stack Navigation.

El stack Navigation contiene varios nodos que, trabajando conjuntamente, permitirán navegar en un entorno cerrado.

El primer nodo del que se va a hablar es map_server. Este nodo tiene la función de guardar el mapa que crea el nodo gmapping y volver a leerlo cuando sea necesario. Al guardar el mapa producirá dos archivos, uno con la imagen creada y el otro es un archivo de configuración, en el cual se podrá cambiar la resolución del mapa, el tamaño, etc.

Una vez obtenido el mapa, configurado de la manera que mejor convenga y cargado, se utilizará el nodo AMCL. Este nodo utiliza el método probabilístico de localización adaptativa Monte Carlo para posicionar al robot en el mapa cargado.

La primera vez que sea cargado se le deberá dar una posición aproximada de salida alrededor de la cual buscará su posición exacta. Esto se puede hacer de varias formas, que se explican en el siguiente capítulo, que es un manual de usuario para trabajar con la aplicación.
Una vez empiece a moverse recalculará la posición en la que se encuentra tomando como referencia los datos que proporciona el robot a través de la odometría, y ajustará la situación en la que realmente se encuentra comparando de nuevo los datos recibidos del láser con el mapa. De esta manera se obtendrá una nube de puntos en los que estima que se puede estar. Cuanto más pequeña sea la nube de puntos más fiables serán los datos sobre la posición.

6.5 Path_follower.

Este nodo es el encargado de planificar la ruta a seguir desde el punto en el que ha situado al robot el nodo anterior, evitando los obstáculos gracias a la información que proporciona el mapa.

Para ello tendrá en cuenta la información que se deduce del mapa, la posición en la que se encuentra inicialmente y la trayectoria que se le cargue. Esta trayectoria constará de uno o varios puntos, con coordenadas, orientación, velocidad y tipo de maniobra que debe realizar, por los que el robot debe pasar.

Se puede observar en la figura 6.6 un ejemplo de trayectoria.

```
  path:
    - {maneouver: 30, node_id: 0, node_type: 1, r_vel: 1, t_vel:
0.5, theta: -0.558, x: 12.507, y: 5.048}

    - {maneouver: 30, node_id: 0, node_type: 1, r_vel: 1, t_vel:
0.5, theta: -0.591, x: 12.511, y: 5.045}

    - {maneouver: 30, node_id: 0, node_type: 1, r_vel: 1, t_vel:
0.5, theta: 1.569, x: 12.696, y: 6.082}

    - {maneouver: 30, node_id: 0, node_type: 1, r_vel: 1,
t_vel: 0.5, theta: -0.494, x: 13.364, y: 5.326}

    - {maneouver: 30, node_id: 0, node_type: 1, r_vel: 1, t_vel:
0.5, theta: 2.546, x: 11.663, y: 6.188
}
```

Figura 6.6. Puntos de la trayectoria

En esta trayectoria se pueden apreciar cinco puntos. Para cada punto está especificada el tipo de maniobra (en este caso para todas es 30), la velocidad máxima con la que hay que pasar por ese punto, la orientación y las coordenadas del punto. En este caso el resto de parámetros no interesan. Están pensados para la aplicación de este nodo a otros robots.
Una vez calculada la trayectoria hasta el siguiente punto de destino le enviará repetidamente instrucciones al nodo Eusebio con la dirección que debe seguir en cada momento para efectuar correctamente la trayectoria.

A continuación le mandará la dirección a seguir al nodo Eusebio. Dicha dirección se le indicará con un mensaje que emula los pulsadores y sticks del joystick para control manual. De manera que tanto si el robot navega de forma autónoma, como si lo hace de forma manual, el nodo Eusebio tenga que interpretar la misma información.

En la figura 6.7 se muestra un ejemplo del mensaje que se envía de este nodo al nodo Eusebio.

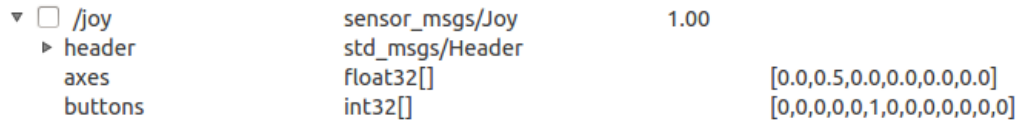


Figura 6.7. Mensaje recibido por el nodo Eusebio.

Como se puede observar, uno de los botones está pulsado. Este es el llamado "pulsador de hombre muerto", si no está pulsado el robot no se desplazará. También se puede observar que se marca una dirección del stick a mitad de velocidad, ya que estos sensores miden entre cero y uno.

6.6 Eusebio.

El nodo Eusebio recibe el mensaje que se acaba de mostrar en la figura 6.7. Con esta información se podrá deducir la dirección que debe seguir el robot y la velocidad con la que debe hacerlo.

Sin embargo para poder seguir la dirección y velocidad fijadas es necesario transformar esta información en las velocidades que debe tomar cada motor. El nodo Eusebio se encarga de esta tarea. Calcula la velocidad que deben tomar los motores del robot para adoptar la dirección y velocidad que permitirán alcanzar la posición objetivo.

La información que llega a este nodo va cambiando, ya que al moverse cambiará la posición y por ello los nodos de más alto nivel recalcularán la posición, la ruta y cambiarán las ordenes que le envían a este nodo. De este modo las velocidades de los motores también variarán.

Una vez calculadas las velocidades y sentidos que debe seguir cada motor en cada momento, éstas serán enviadas al nodo Gpmrc_node en un mensaje como éste como el mostrado en la figura 6.8.

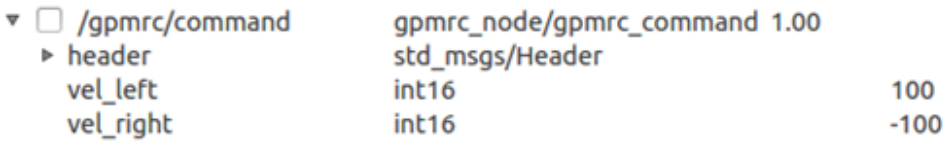


Figura 6.8. Mensaje enviado a Gpmrc_node.

En este caso se está indicando que tiene que adoptar una velocidad de 100 en ambos motores de un máximo de 2000. Además los signos indican que ambas ruedas deben girar de manera que el robot avance en línea recta ya que está programado de esa manera.

6.7 Gpmrc_node.

Este nodo recibirá mensajes del nodo Eusebio, en los cuales vendrá especificada la velocidad y el sentido que ha calculado que tiene que adoptar cada motor para seguir la dirección y velocidad deseadas. Una vez recibido el mensaje, este nodo se encargará de transformar la información de manera que pueda ser interpretada por la placa GPMRC, la cual enviará la información oportuna a las controladoras de motores.

La información enviada a la GPMRC son secuencias ASCII de manera que el nodo tendrá que convertir la información a tramas ASCII para poder ser interpretadas.

6.8 Serial_port.

Para la comunicación con el robot hay que conectar la placa controladora de bajo nivel GPMRC con el dispositivo de control a alto nivel, en este caso una Raspberry Pi. La función que cumple este nodo es la de actuar como una biblioteca que habilita la comunicación serial entre los dos dispositivos. Por tanto, sin este nodo no funcionaría ninguno de los de niveles superiores.

Capítulo 7:

Manual de usuario.

En este capítulo se desarrollará un manual de usuario básico para saber ejecutar la aplicación que se ha descrito en el apartado anterior así como información importante para comprender cómo ha sido desarrollada el proyecto y que podría ser de utilidad para trabajos futuros basados en él.

7.1 Copias de seguridad.

Se va a comenzar dando unos conocimientos interesantes para el desarrollo de proyectos en la plataforma Raspberry Pi y que se han utilizado para este trabajo.

Es interesante empezar este capítulo con algo que es necesario hacer a menudo a lo largo del desarrollo del proyecto: la creación de copias de seguridad.

Para hacer una copia de seguridad del contenido de la tarjeta SD que está utilizando la Raspberry se debe crear una imagen.

Una imagen es un archivo donde se almacena una copia o imagen exacta de un sistema de ficheros, se utiliza para transferir archivos que necesitan evitar la pérdida de cualquier información o la modificación de la estructura original, necesaria muchas veces para el correcto funcionamiento del programa, éste es el caso. No se puede perder ningún tipo de información, por lo que una copia normal no sirve, sino que hay que hacer una copia mediante imagen.

Para ello hay que seguir los siguientes pasos:

Para comenzar hay que mirar el nombre de la tarjeta SD introduciendo en un terminal:

```
df -h
```

La respuesta obtenida en esta caso es que se llama /dev/sdb, con dos particiones: sdb1 y sdb2.

A continuación se desmontan todas las particiones con las que cuenta la tarjeta:

```
umount /dev/sdb1
umount /dev/sdb2
```

El siguiente paso es crear la imagen. Esto se realizará mediante la siguiente instrucción.

```
sudo dd if=/dev/sdb of=~/Projects/copia_seguridad.img
```

De esta manera se creará una imagen llamada copia_seguridad de todas las particiones de la tarjeta sdb en la dirección ~/Projects.

El proceso puede tardar un tiempo, depende del tamaño de la SD, y no se mostrará por la terminal ningún indicador del tiempo que falta para terminar.

Lo que se ha creado es una "imagen" de la tarjeta SD, lo que significa que como la tarjeta tenía 32 GB capacidad, entonces el tamaño del archivo .img será de 32 GB, independientemente de si la tarjeta estaba llena o vacía de contenido en el momento de la copia.

Si se comprime la imagen en un archivo .zip se obtendrá un archivo con un tamaño más parecido al contenido real de la SD, más fácil de almacenar.

En el caso contrario, para cargar una imagen en la SD habrá que seguir los siguientes pasos, muy parecidos a los de crearla

Igual que en el caso anterior hay que mirar el nombre de la tarjeta y desmontar todas las particiones

```
df -h
umount /dev/sdb1
umount /dev/sdb2
```

A continuación se carga la imagen poniendo como origen (if) el archivo .img almacenado en el ordenador, y como destino (of) la tarjeta sin especificar particiones.

```
sudo dd if=~/Projects/copia_seguridad.img of=/dev/sdb
```

El último paso, que es muy importante, es ejecutar el comando

```
sudo sync
```

para borrar la memoria cache y asegurarse de que se ha grabado en la SD.

7.2 Ejecución de nodos en distintas máquinas.

A continuación se va a explicar algo muy importante para el funcionamiento de la aplicación, la comunicación entre Raspberry y ordenador.

Como se dijo en el anterior capítulo, la aplicación funcionará con el ordenador y la Raspberry Pi trabajando de manera conjunta, ya que habrá nodos en ambas máquinas y deberán enviarse datos entre ellas. En este apartado se expondrá cómo hacer posible la comunicación entre ambos dispositivos.

Entre los dos dispositivos habrá una relación de esclavo y maestro. Por tanto hay que decidir cuál de las dos máquinas va a ser el maestro y comunicárselo a ambas.

Para ello se ejecuta la instrucción:

```
export ROS_MASTER_URI=http://nombredelmaestro:11311/
```

Si no se conoce el nombre de la máquina, éste aparece al introducir el comando "hostname". En este caso la Raspberry Pi será la máquina maestra, y dado que no se ha cambiado el hostname, éste seguirá siendo el que tiene por defecto: raspberrypi.

Las dos máquinas con las que se va a trabajar son: sacarino2 con IP 192.168.107.91 y raspberrypi con IP 192.168.107.90.

Hay que tener en cuenta que antes de ejecutar los siguientes comandos que hay que modificar los ficheros "hosts" dentro de la carpeta etc para que reconozcan a una maquina por su hostname y no por su IP.

Para conseguir eso debe abrirse el fichero hosts con permisos de administrador y añadir una línea: "192.168.107.90 raspberrypi" en el fichero de sacarino2 y "192.168.107.91 sacarino2" en el fichero que contiene la Raspberry.

Como ejemplo de conexión se van a ejecutar dos nodos que se utilizan para los tutoriales de ROS. Estos nodos son “talker”, que envía información a través de un topic, y “listener”, que se suscribe al topic creado por talker y recibe los mensajes que por éste circulan.

Para hacer que la Raspberry sea la maquina maestra se seguirán los siguientes pasos:

En la raspberrypi:

1- Ejecutar el comando roscore en un terminal

2- En un segundo terminal introducir

```
export ROS_MASTER_URI=http://raspberrypi:11311/
```

3- Ejecutar el nodo. En este caso talker:

```
rosrun beginner_tutorials talker
```

En sacarino2:

4- Comunicar al ordenador que el maestro va a ser raspberrypi. En esta máquina no se ejecutará un roscore, ya que se utilizará solamente uno, que será el de la máquina maestra.

```
export ROS_MASTER_URI=http://raspberrypi:11311/
```

5- En esa misma terminal ejecutar el nodo que se comunicara con el que se está ejecutando en la Raspberry. En este caso listener:

```
rosrun beginner_tutorials listener
```

Esto va a funcionar siempre y cuando se haya modificado el fichero "hosts" en la carpeta etc de la maquina esclava.

Para hacer una comunicación bidireccional (es decir, que sacarino2 pueda actuar de maestro) se debe modificar el fichero hosts de las dos máquinas.

También hay que tener en cuenta que al conectarse por Ethernet o por WIFI las direcciones IP que utiliza cada dispositivo no son las mismas por lo que se deben incluir en los archivos hosts los dos casos o modificarlos cuando se modifique el tipo de conexión.

7.3 Lanzamiento de los nodos de navegación.

En este apartado se explicará todo lo necesario para, partiendo del punto en el que se terminó en los anteriores capítulos, poder ejecutar los nodos que harán posible la navegación.

7.3.1 Compilación.

Antes de poder ejecutar los nodos lo primero que hay que hacer es compilarlos.

Para ello debe hacerse en primer lugar un espacio de trabajo de catkin, que es el sistema de compilación que va a utilizarse. En el apartado 5.2.1 de este proyecto se explica cómo hacerlo.

Una vez creado el espacio de trabajo hay que guardar en la carpeta src los nodos a compilar. Estos nodos serán carpetas que deben contener ficheros MakeLists y Package, además de ficheros .cpp .h .msg etc. En la versión digital de este proyecto se incluyen dichos archivos.

Una vez guardados, en el terminal hay que desplazarse a la carpeta espacio de trabajo (en este caso Proyecto_eusebio) y compilar con el comando catkin_make.

Ya que Raspberry Pi tiene poca capacidad de procesamiento tardará un tiempo en compilar.

Una vez compilados hay que dar permisos antes de ejecutar los programas.

7.3.2 Cambios de permisos.

Antes de lanzar los nodos hay que configurar el puerto USB para darle más permisos ya que sino no se podrá conectar al robot. Para conseguir la conexión en primer lugar deben conectarse todos los elementos necesarios y a continuación seguir las siguientes instrucciones:

Mirar el nombre del USB ejecutando el comando

```
lsusb
```

Este comando mostrará una lista de los dispositivos conectados. En este caso es:

> Bus 002 Device 016: ID 0403:6001 Future Technology Devices International, Ltd FT232 USB-Serial (UART) IC

A continuación hay que ejecutar esta instrucción.

```
sudo modprobe usbserial vendor=0x0403 product=0x6001
```

Habrá que fijarse en el ID que mostraba el dispositivo en la instrucción anterior y cambiar los números por los correspondientes en cada caso.

Por último hay que ejecutar la siguiente instrucción para conceder más permisos para la conexión USB del robot y del láser. En este caso:

Para la conexión con el robot:

```
sudo chown raspberrypi /dev/ttyUSB0
```

Para la conexión con el láser:

```
sudo chown raspberrypi /dev/ttyACM0
```

ttyUSB0 y ttyACM0 son los nombres que adoptan los dispositivos en este caso. Para averiguar cómo se llaman en otro caso se debe ejecutar el comando **df –h**.

7.3.3 Lanzamiento de nodos.

Ahora ya pueden para lanzarse los nodos. En este caso se hará mediante un método sencillo, a través de los ficheros .launch.

Los ficheros .launch buscan un roscore ejecutándose y si no lo encuentran lo lanzan ellos. De la misma manera, buscan si están ejecutándose los nodos que tiene incluidos, y si no lo están haciendo los lanza.

Por ejemplo se puede lanzar el nodo gprmc_node a través del fichero llamado gpmrc_node.launch que se encuentra dentro de la carpeta launch, contenida en la carpeta gpmrc_node.

Para ilustrar sobre el contenido de este tipo de ficheros se muestra el contenido de gpmrc_node.launch:

```
<launch>
   <!-- ODOMETRY -->
  <node pkg="gpmrc_node" type="gpmrc_node" name="gpmrc_node" cwd="node" args="-port /dev/ttyUSB0 -baud 38400" respawn="true" output="screen"> </node>

</launch>
```

Este fichero se encargará de ejecutar todos los nodos que tiene incluidos en sus líneas, y con ello se conseguirá que el nodo gpmrc_node funcione adecuadamente.

Para lanzar el nodo gpmrc_node hay que ejecutar la siguiente instrucción:

```
roslaunch gpmrc_node gpmrc_node.launch
```

Para ejecutar todos los nodos que componen la aplicación se debe lanzar el fichero Eusebio.launch en la Raspberry Pi, mientras que en el ordenador habrá que indicar que la máquina maestra será la Raspberry Pi y a continuación lanzar el archivo eusebio_remote_map.launch.

Hay que recordar que al final de este proyecto la Raspberry Pi debe ir integrada dentro del robot, por lo que la única manera de lanzar los nodos en ella es conectándose de manera remota. Esto ya se explicó cómo hacerlo en el capítulo 4 de este proyecto.

7.3.4 Comunicación con el robot.

En este punto se podrá ver qué datos está transmitiendo el robot, y se podrá enviar instrucciones de movimiento desde el ordenador o bien con el control manual.

Por ejemplo para ver los mensajes del estado en el que se encuentra el robot debe abrirse la herramienta rqt, y dentro de ella la opción Topic monitor.

Al buscar el topic /gpmrc/state podrá verse lo que se está enviando por ese topic en cada momento. Se puede apreciar un ejemplo en la imagen 7.1.

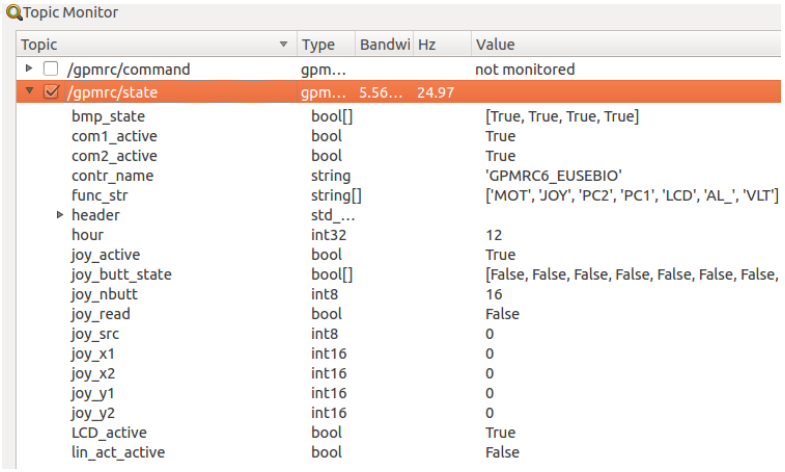


Figura 7.1. Visualización de datos del robot a través de rqt.

Se pueden mandar mensajes al robot a través de la herramienta rqt.

Para lanzarla se ejecutará en una terminal la instrucción **rqt.** Una vez abierto debe ponerse en topic /gpmrc/commands y en type gpmrc_node/gpmrc_command para mandar directamente las velocidades de los motores, o bien /joy y type sensor_msgs/Joy si lo que se desea es simular que los datos que se envían con el joystick.

En la imagen 7.2 se puede ver un ejemplo del envío de órdenes a través de la herramienta rqt.

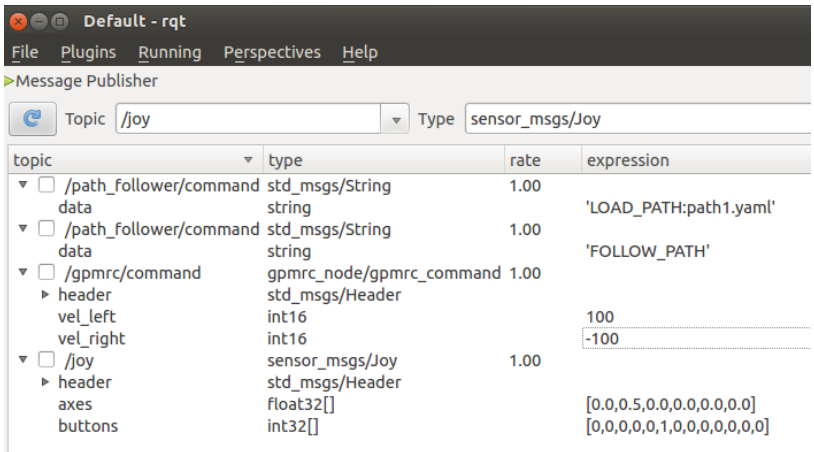


Figura 7.2. Envío de mensajes a través de la herramienta rqt.

Con este apartado se conocen todos los pasos a seguir para poner en funcionamiento la aplicación.

Capítulo 8:

Validación experimental

La finalidad de este capítulo es hacer una retrospectiva a los objetivos iniciales del proyecto y comprobar si estos se han cumplido.

Para que sea más sencilla la visualización de los resultados se volverán a enumerar los objetivos del proyecto. Estos son:

- Selección de una plataforma de control a alto nivel que solvente la problemática del anterior sistema.

- Selección del sistema operativo y de la distribución que utilizará el dispositivo de control.

- Incorporación de un sistema de navegación para que el robot se localice y sepa desplazarse esquivando los posibles obstáculos en un entorno memorizado previamente.

- Realización de todos los objetivos anteriores con un coste moderado.

El primer objetivo se cumplió cuando en el capítulo tres de este proyecto se seleccionó como método de control de alto nivel la combinación de Raspberry Pi integrada en la plataforma y un ordenador con el que se controlará y supervisará de forma remota el funcionamiento del robot.

En la imagen 8.1 se puede apreciar la plataforma de control integrada en la plataforma. En contraste con ella se puede ver la plataforma de control anterior (PC), que ocupaba mucho más espacio, tenía un mayor peso y era más vulnerable que el sistema ahora elegido.

Figura 8.1. Comparativa de sistemas de control.

En la figura anterior se ha colocado la Raspberry Pi encima de la plataforma para visualizar la diferencia entre los dos métodos, sin embargo, pueden conectarse todos los elementos por dentro de la estructura, de manera que la parte superior de la plataforma robótica quede completamente libre (figura 8.2).

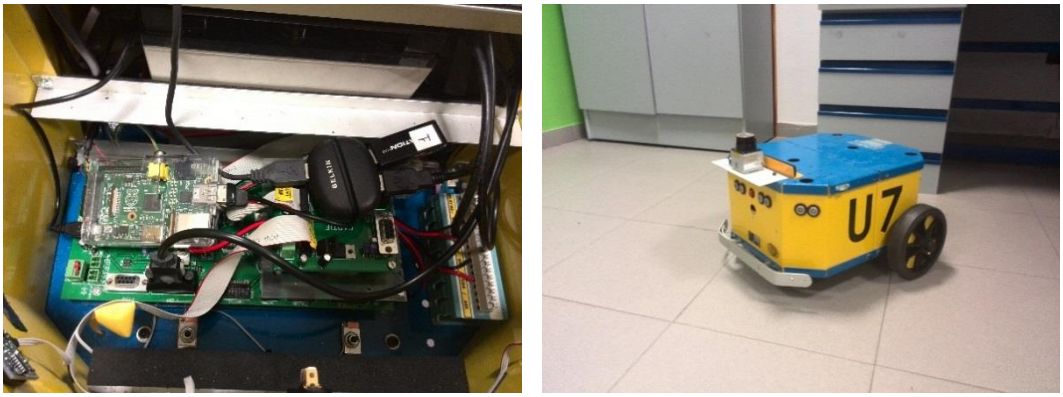


Figura 8.2. Sistema de control actual incluido en la estructura.2

También se seleccionó en el capítulo tres que la distribución de sistema operativo que se utilizaría sería Raspbian y además se optó por desarrollar el software de control sobre un framework, ROS (Robot Operating System), que facilita librerías, programas y herramientas sobre las que desarrollar aplicaciones robóticas, por lo que el segundo requisito queda satisfecho.

Como tercer objetivo se proponía la navegación autónoma del robot, de manera que este se localizase y fuese capaz de desplazarse sin chocar de un punto a otro elegido en un entorno conocido.

Para cumplir con este objetivo se desarrolló una aplicación dividida en varios módulos, los cuales se explicaron en el capítulo seis, donde se detalla la función de cada uno.

En esta parte del desarrollo del proyecto se realizaron numerosas pruebas para comprobar el correcto funcionamiento de los diferentes nodos. Empezando por los nodos a más bajo nivel (gpmrc y serial_port) hasta el más alto nivel, donde se realizan los mapas, el posicionamiento del robot dentro de ellos y se calcula la trayectoria para llegar hasta el punto de destino.

En primer lugar se comprobó el buen funcionamiento de los nodos Gpmrc_node y Serial_port. Esto se consigue simulando las señales que llegan desde el nodo Eusebio hasta la gpmrc. Para ello se utilizará la herramienta de ROS rqt. Habrá que suscribirse al topic gpmrc/command y enviarle las velocidades y direcciones que han de seguir los motores.

En la figura 8.3 se puede apreciar cómo se realiza el envío de instrucciones mediante la herramienta rqt.

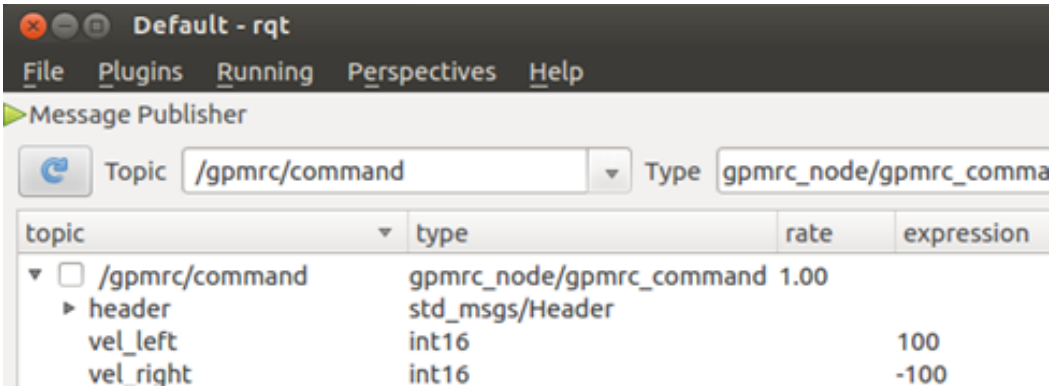


Figura 8.3. Envío de mensajes tipo gpmrc/command.

Para la realización de estas pruebas se colocó el robot sobre una plataforma, de manera que se observase el movimiento de los motores sin desplazamiento del robot.

De la misma manera se realizaron pruebas de funcionamiento para el nodo eusebio, simulando la llegada de mensajes tipo joy, y comprobando la respuesta de los motores (figura 8.4).

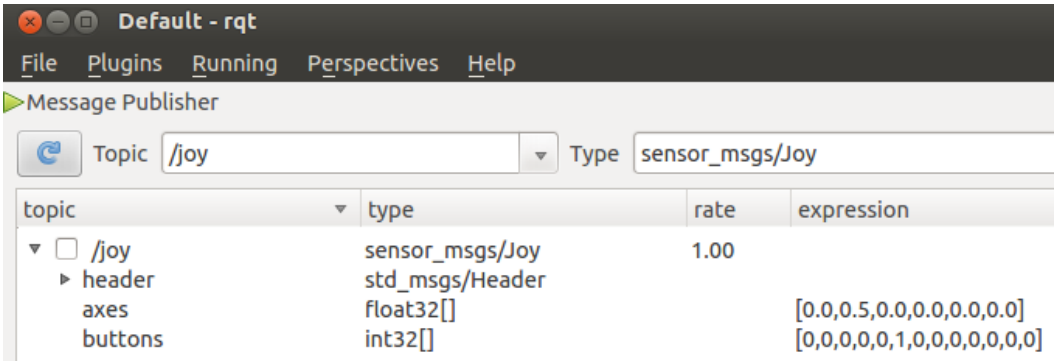


Figura 8.4. Envío de mensajes tipo joy.

Una vez que estos nodos funcionaron correctamente se pasó a los que controlan la localización, mapeo y sorteo de obstáculos del robot.

Para comprobar el funcionamiento del nodo Hokuyo simplemente se ejecutará por separado y nos suscribiremos al topic por el que envía la información que recibe el sensor. Entonces aparecerá por pantalla algo similar a lo que aparece en la siguiente imagen 8.5.

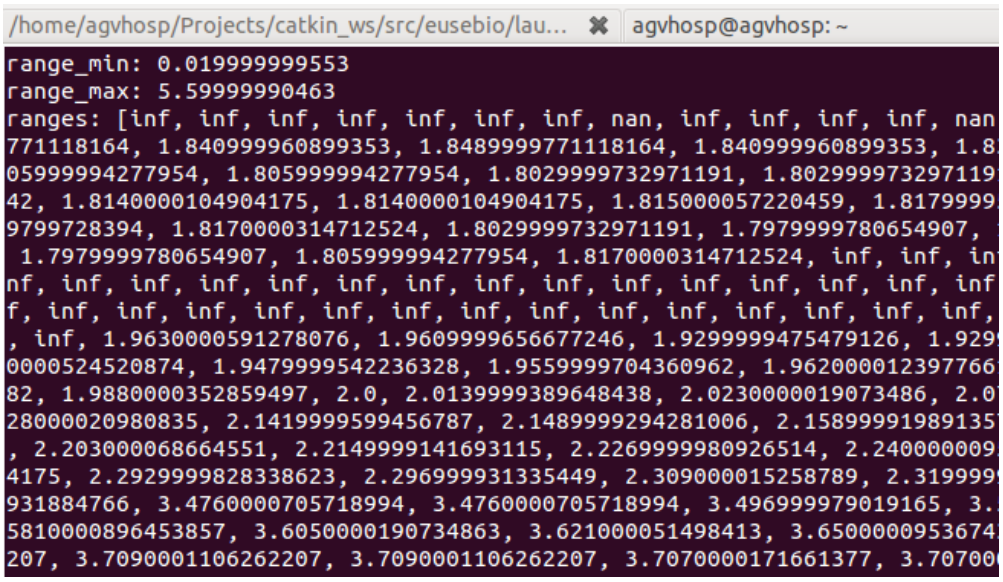


Figura 8.5. Lectura de los datos proporcionados por el láser.

De esta forma se comprueba que el sistema laser está funcionando, sin embargo no puede saberse si lo hace correctamente. Para ello es necesario representar la información. De esto se encargará la herramienta rviz, integrada en ROS, que dibujará gráficamente los datos que proporciona el sensor. Si se lanza el nodo Hokuyo y se abre la herramienta de ROS rviz debería aparecer algo como lo que puede verse en la figura 8.6:

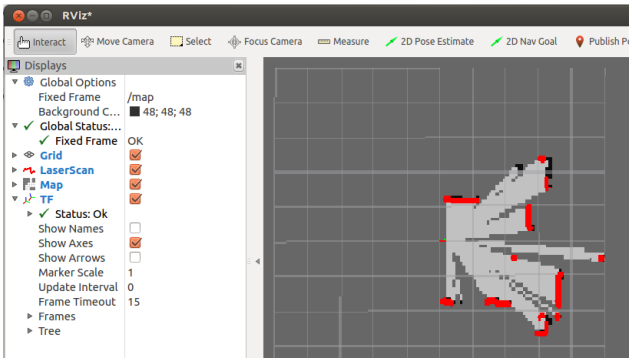


Figura 8.6. Representación de los datos gráficamente.

Las líneas rojas son los obstáculos que está detectando el robot en ese momento. Cuando los obstáculos permanecen en un mismo sitio el nodo gmapping los remarca en negro, de manera que al desplazarse el robot, se creará un mapa de obstáculos permanentes.

Para comprobar también el funcionamiento del nodo gmapping se intentará crear un pequeño mapa de los obstáculos más cercanos. Este nodo tiene en cuenta los datos que le proporciona el láser y la odometría del robot, de manera que al desplazarse el robot, tiene en cuenta las distancias y giros, y es por ello que se puede realizar un mapa en todas las direcciones de un espacio relativamente grande.

Dicho esto se procederá a crear un mapa del espacio más cercano tal y como se explicó en el capítulo 7, Manual de usuario.

El resultado de esta prueba es la que se muestra en la figura 8.7.

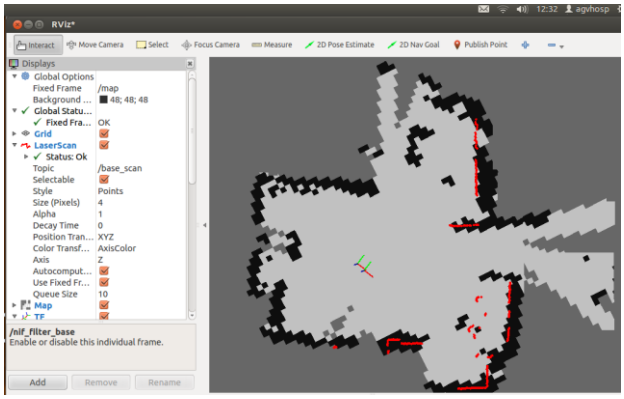


Figura 8.7. Mapeado de una zona pequeña.

Tanto en la figura 8.6 como en la 8.7 la orientación del robot es la misma.

Una vez visto esto habrá que comparar con el espacio real y la orientación del robot (figura 8.8).

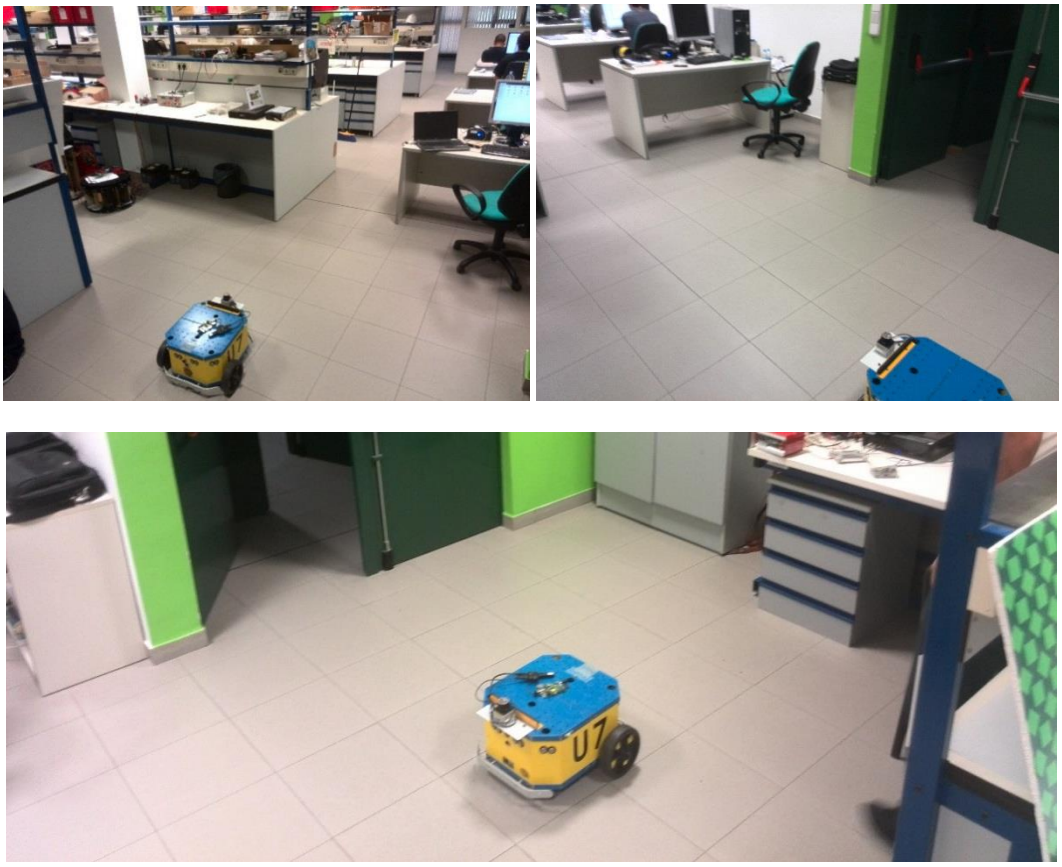


Figura 8.8. Entorno real del robot.

La imagen que creada se corresponde con los objetos alrededor del robot por lo que queda comprobado que estos dos nodos funcionan correctamente.

Para que al desplazarse el robot dibuje el mapa correctamente, es necesario haber calibrado la odometría, es decir, la estimación de la posición durante la navegación. Para realizar esta estimación se usa información sobre la rotación de las ruedas.

Es por esto que se dará aquí una explicación de cómo se ha realizado la calibración de la odometría.

En primer lugar hay que colocar al robot en una posición de inicio a partir de la cual pueda medirse el recorrido fácilmente. Hecho esto se debe desplazar el robot hacia delante hasta que se crea conveniente. En este caso se ha decidido que con desplazar el robot aproximadamente un metro será suficiente. Una vez el robot se detenga, hay que medir con cinta métrica cuánto se ha desplazado.

En la figura 8.9 puede verse el robot preparado para la medida del desplazamiento.

Figura 8.9. Robot en posición inicial.

A continuación se ejecuta el movimiento y se para cuando haya recorrido alrededor de un metro.

El siguiente paso será visualizar mediante el Topic monitor, opción de la herramienta rqt, el topic Robot_Status, que, entre otras cosas, proporciona información sobre el desplazamiento del robot (figura 8.10).

En este caso muestra lo siguiente:

Topic Monitor

Topic	Type	Bandwidth	Hz	Value
/eusebio/RobotStatus	eusebio/RobotStatus	1.07KB/s	1.93	
steering_ang	float32			0.0
theta	float32			-0.11622608453035355
total_distance_travelled	float32			5.2174177169799805
total_moving_time	float32			65.56050872802734
total_working_time	float32			2267.647705078125
xc	float32			0.0
xs	float32			-0.0044230236671864 99
yc	float32			0.0
ys	float32			-0.09342482686042786
partial_working_time	float32			448.2337951660156
position_calibrated	bool			False
ref_steer	float32			0.0
ref_vel	float32			-0.5
robotID	string			'EUSEBIO-V1.0'
sec	int32			17

Figura 8.10. Desplazamiento realizado según el robot.

Como puede verse, el robot indica que se ha desplazado 93.42 cm en la dirección del eje y, y 4.42 cm en la dirección del eje x.

Si se mide lo que se ha desplazado realmente se ve que se ha desplazado 93.5 cm en el eje y, desde el eje central del robot, y que efectivamente se ha desplazado en el eje x a pesar de ser una operación de desplazamiento en línea recta. En la imagen 8.11 se puede ver la medida real del desplazamiento. Mientras que en la figura 8.12 puede apreciarse el desvío hacia la izquierda

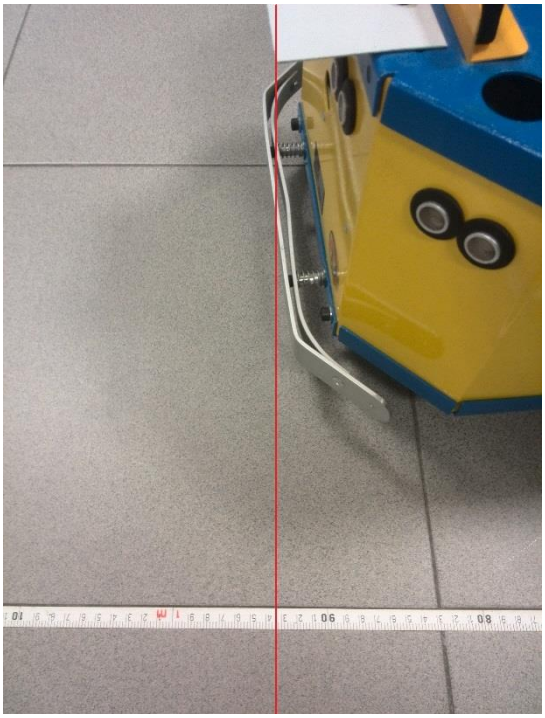


Figura 8.11. Desplazamiento real.

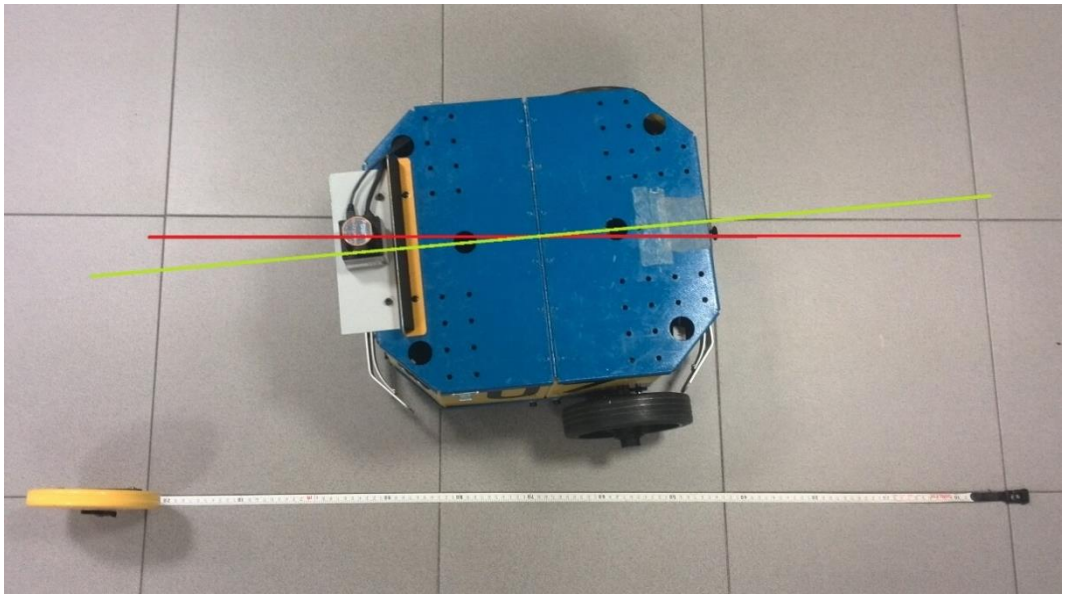


Figura 8.12. Desvío de la trayectoria.

En este caso el desplazamiento se corresponde de manera bastante exacta, por lo que no hace falta realizar modificaciones, sin embargo, si hiciese falta habría que modificar el fichero robot_parameters.yaml. Que se encuentra en la carpeta conf dentro del nodo Eusebio. En este archivo aparecen varios datos de configuración. Entre ellos está PlsDist, (que en este caso es 1825.71), este número es el que efectúa el cambio entre el número de pulsos que se reciben del encoder y la distancia recorrida.

Si la distancia que calcula el robot que ha recorrido no coincide con la recorrida realmente deberá modificarse este número hasta que coincidan.

Por último, para comprobar el posicionamiento en el mapa y la navegación de un punto a otro esquivando los obstáculos se ejecutarán conjuntamente todos los nodos que conforman la aplicación y que se enumeran en el capítulo seis.

Con ayuda de la herramienta rviz se comprueba si los ejes, que se sitúan en el centro de coordenadas del robot, están correctamente situados y orientados.

A continuación se moverá manualmente el robot y se comprobará que el eje de coordenadas se mueva de la misma forma que el robot en la realidad.

Para verificar el funcionamiento de la navegación se carga una trayectoria mediante la herramienta rqt y se ejecuta. A continuación se abre el topic path_follower/command y se escribe 'LOAD_PATH:nombre_trayectoria.yaml' para cargar la trayectoria descrita en el archivo nombre_trayectoria.yaml y a continuación se sustituye la instrucción por 'FOLLOW_PATH' para empezar a seguir la trayectoria. Estas instrucciones pueden verse en la figura 8.13.

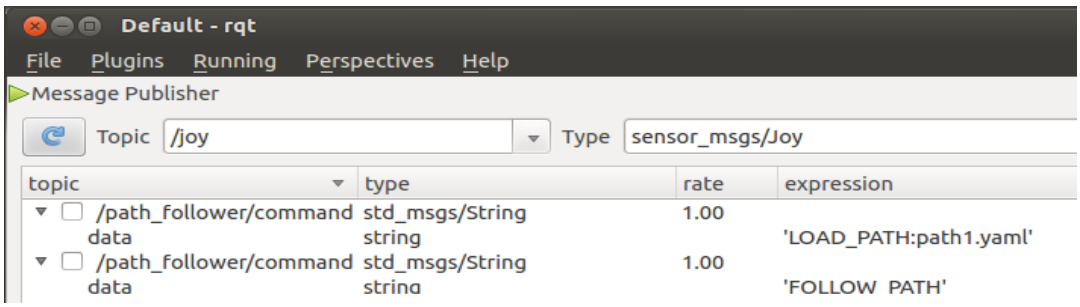


Figura 8.13. Instrucciones para seguir una trayectoria.

En este caso se ha conseguido cumplir las pruebas con éxito, de manera que puede decirse que se ha cumplido el tercer objetivo del proyecto.

El último objetivo, realizar el proyecto con un coste moderado, aún no puede justificarse numéricamente, dado que aún no se ha realizado el presupuesto. Sin embargo, los elementos que se han añadido a la plataforma inicial son pocos y el software que se ha utilizado es de libre desarrollo por lo que se puede prever que el coste final del proyecto no será muy elevado.

Con esto puede decirse que los objetivos del proyecto se han cumplido.

Capítulo 9:

Estudio económico

En este apartado se estimará el coste del desarrollo de este proyecto a la vez que se enumerarán los gastos asociados a los materiales de los que está formada la plataforma robótica. Los costes serán agrupados según el origen y divididos en costes directos e indirectos. Por último se detalla el presupuesto total.

9.1 Costes directos.

Los costes directos son aquellos que pueden identificarse con un producto o servicio, por lo que este apartado contendrá el importe de los costes de material, construcción, recursos y herramientas empleadas durante el desarrollo del robot.

9.1.1 Costes asociados al equipo.

- **Costes de material**

En primer lugar se expondrán los costes del material necesario para la construcción del robot. Dichos costes pueden verse en las tablas 9.1, 9.2 y 9.3. Por último se realiza una tabla resumen del coste total que suponen los costes de material. Dicha tabla es la 9.4.

Elementos Mecánicos				
Concepto	Procedencia	Coste unitario (€)	Unidades	Coste total (€)
Base del chasis	Talleres Tejedor	20.00	1	20.00
Tapa abisagrada	Talleres Tejedor	15.00	2	30.00
Paredes del chasis	Talleres Tejedor	20.00	2	40.00
Buje de rueda	Talleres Tejedor	45.00	2	90.00
Rodamientos rueda SFK 6003 12,DE,NC	Rodamientos Palencia	13.33	2	26.66
Rueda 170x32	Ruedas ALEX	22.60	2	45.20
Ruedas locas Caster esfera 40	Ruedas ALEX	4.50	2	9.00
Bumpers	Service Robotics	35.80	2	71.60
Mecanismo paragolpes	Service Robotics	2.50	4	10.00
Ángulo protección	Service Robotics	3.00	1	3.00
Cierres tapa	Ferretería Pablo Llamas	1.80	2	3.60
ISO 7380 -M6 X 16	Ferretería Pablo Llamas	0.15	34	5.10
DIN 912 M4 x 20	Ferretería Pablo Llamas	0.16	6	0.96
DIN 912 M5 x 10	Ferretería Pablo Llamas	0.15	4	0.60
DIN 7991 M5 x 20	Ferretería Pablo Llamas	0.15	4	0.60
DIN EN ISO 10511 M6	Ferretería Pablo Llamas	0.10	1	0.10
Autorroscante DIN 7982 - 9 x 9.5	Ferretería Pablo Llamas	0.05	2	0.10
Arandela DIN 433 - 5.3	Ferretería Pablo Llamas	0.02	4	0.08
Arandela ISO 7090 - 8	Ferretería Pablo Llamas	0.02	8	0.16
Remacha M5 x 10	Ferretería Pablo Llamas	0.02	4	0.08
			Total	356.84

Tabla 9.1. Coste de los elementos mecánicos del robot.

Elementos Eléctricos				
Concepto	Procedencia	Coste unitario (€)	Unidades	Coste total (€)
Batería Yasa 14AHR 12V PB	Solo Recambios	42.00	1	42.00
Conector de carga XLR Base	Electrosón Castilla	7.50	1	7.50
Portafusibles 4x4	Soto Recambios	19.91	1	19.91
Canaleta 25x25 Bornes	Electrosón Castilla	0.80	1	0.80
Borna 3 conectores	Electrosón Castilla	0.50	6	3.00
Interruptor general	Electrosón Castilla	1.85	1	1.85
Cableado	Electrosón Castilla	0.71	4	2.84
Conector servo	Electrosón Castilla	1.43	1	1.43
			Total	79.33

Tabla 9.2. Coste de los elementos eléctricos del robot.

Elementos Electrónicos				
Concepto	Procedencia	Coste unitario (€)	Unidades	Coste total (€)
Controladora GPMRC6LC	Centro tecnológico CARTIF	320,00	1	320,00
Lector de sonars	Centro tecnológico CARTIF	110,00	1	110,00
Motor DC-x42x550	Superdroid Robots	54,00	2	108,00
Encoder 12PPR 42 x 15	Superdroid Robots	32,60	2	65,20
Controlador de motores	Centro tecnológico CARTIF	70,00	2	140,00
Driver para controlador de motores	Centro tecnológico CARTIF	50,00	2	100,00
Sonar TS-601Audiowell	Audiowell Electronic	7,00	8	56,00
Interruptor Tact SMD Bumper	Electrosón Catilla	0,50	4	2,00
Mando tipo PlayStation y receptor	Superrobotica	24,85	1	24,85
Cable plano múltiples vías	Electrosón Catilla	2,90	3	8,70
Adaptador RS232-USB	Prolific Technology	20,00	1	20,00
Raspberry Pi	Farnell España	26,06	1	26,06
Laser	Robotshop	1680,00	1	1680,00
			Total	2755,81

Tabla 9.3.Coste de los elementos electrónicos del robot.

Coste Material del robot		
Concepto		Coste (€)
Elementos mecánicos		356,84
Elementos eléctricos		79,33
Dispositivos electrónicos		2755,81
	Suma	3191.98
	IVA 21%	670,32
	Total	**3862.30**

Tabla 9.4. Costes del material para el robot Eusebio.

- ## Costes de herramientas y equipo.

Se incluirán en este apartado los costes derivados del uso de herramientas y el equipo necesario para la construcción física del robot y para la implementación del sistema de control empotrado. Estos costes se detallan en la tabla 9.5.

Coste de herramientas y equipo utilizado			
Concepto	Coste unitario (€/h)	Unidades (horas)	Coste total (€)
Banco de trabajo (herramientas varias)	2,32	10	23,20
Material de laboratorio (polímetro, soldador, etc)	0,75	20	15,00
Dispositivos electrónicos (fuente de alimentación, osciloscopio, generador de señal)	0,93	20	18,60
Ordenador de sobremesa	0,29	600	174,00
		Total	**230,80**

Tabla 9.5. Costes de herramientas y equipo.

- ## Costes de software.

En este caso se han utilizado varios software, todos ellos de libre distribución, por lo que el coste total del software será cero. En la tabla 9.6 se pueden ver además de los costes de software, una estimación de las horas que ha sido utilizado cada uno.

Coste del software empleado			
Concepto	Coste unitario (€/h)	Unidades(h)	Coste total (€)
Ubuntu	0,00	600	0,00
Raspbian	0,00	200	0,00
ROS	0,00	180	0,00
		Total	**0,00**

Tabla 9.6. Coste de software.

- **Resumen de costes asociados al equipo.**

En la tabla 9.7 pueden verse los costes asociados a cada apartado y su suma.

Resumen de costes asociados al equipo	
Concepto	**Importe (€)**
Coste material	3862.30
Herramientas y equipo utilizado	230,80
Software empleado	0,00
Total	**4093.10**

Tabla 9.7. Suma de costes asociados al equipo.

9.1.2 Coste de la mano de obra por tiempo empleado.

En este apartado se va a calcular el coste de la mano de obra necesaria para la puesta en marcha del prototipo diseñado. Se incluirá en primer lugar el coste del montaje de los diferentes componentes mecánicos, eléctricos y electrónicos del robot. Este trabajo podrá ser realizado por un titulado en formación profesional, cuya mano de obra es más barata que la de un ingeniero.

En primer lugar se calcularán las horas de trabajo que hay en un año, de esta manera podrá calcularse el sueldo por horas y ajustar el pago al tiempo que ha sido necesario para realizar cada parte del trabajo. El cálculo se detalla en la tabla 9.8.

Coste de la mano de obra por tiempo empleado	
Concepto	**Tiempo (días)**
Año medio	365
Vacaciones	22
Días festivos	14
Días perdidos (aprox.)	5
Fines de semana	104
Total	**220**
x 8 horas/día	**1760 h laborables/año**

Tabla 9.8. Cálculo de las horas laborables en un año.

Una vez calculadas las horas laborables por año se tasará el sueldo que cobran un ingeniero y un técnico superior de formación profesional a la hora, teniendo en cuenta el sueldo medio de estos profesionales en España con un nivel medio de experiencia. El cálculo puede verse en las tablas 9.9 y 9.10.

Cálculo del suelo de un ingeniero por hora	
Concepto	**Importe (€)**
Sueldo bruto e incentivos	27600,00
Sueldo neto e incentivos	25806,00
S.S. a cargo de la empresa	8197,00
Sueldo Total	**35797,00**
Sueldo por hora	**20,34**

Tabla 9.9. Cálculo del sueldo por horas de un ingeniero.

Cálculo del sueldo de un técnico por hora	
Concepto	**Importe (€)**
Sueldo bruto e incentivos	17500,00
Sueldo neto e incentivos	16362,50
S.S. a cargo de la empresa	5197,50
Sueldo Total	**22697,50**
Sueldo por hora	**12.90**

Tabla 9.10. Cálculo del sueldo por horas de un técnico.

Una vez conocido el sueldo por horas se estimarán las horas necesarias que ha de realizar cada trabajador y con ello se estimarán los costes totales de personal.

El tiempo de desarrollo del proyecto ha sido de cinco meses. Antes se ha calculado que en un año las horas laborables son 1760, por lo que las horas laborables en 5 meses trabajando 8 horas diarias serán 740.

Por otro lado el tiempo de montaje del robot se estima que son unas 16 horas, por lo que el coste total de personal se refleja en la tabla 9.11.

Costes de personal			
Concepto	**Coste unitario (€)**	**Unidades**	**Coste Total (€)**
Construcción del robot	12.90	16.00	206.40
Desarrollo del proyecto	20.34	740.00	15254.40
		Total	**15460.80**

Tabla 9.11. Costes de personal.

9.1.3 Otros costes directos.

Se incluyen en este apartado los costes directos que no pueden atribuirse a elementos o herramientas para la construcción del robot ni a costes de personal. Irán incluidos en este apartado los gastos por ejemplo del material de oficina, tal como folios, fotocopias, etc. y material de laboratorio, como cinta adhesiva o estaño. También se incluirán los costes para la documentación de este proyecto. Puede verse el detalle de estos costes en la tabla 9.12.

Otros costes directos	
Concepto	**Importe (€)**
Material de laboratorio	50
Material de oficina	30
Documentación del proyecto	60
Total	**140**

Tabla 9.12.Otros costes directos.

9.1.4 Total de costes directos.

En la tabla 9.13 que se expondrá a continuación puede verse un resumen de los gastos que suponen los costes directos.

Total de costes directos	
Concepto	**Importe (€)**
Costes asociados al equipo	4093.10
Coste de personal directo	15460.80
Otros costes	140.00
Total	**19693.90**

Tabla 9.13.Total de costes directos.

9.2. Costes indirectos.

Los costes indirectos son aquellos que corresponden a recursos consumidos durante su realización pero que no pueden asociarse directamente a la realización de este proyecto. Dentro de estos gastos se incluyen por ejemplo los gastos de electricidad, agua, climatización, etc. Haciendo un cálculo estimado se obtendrán como resultado los costes que pueden apreciarse en la tabla 9.14.

Costes indirectos	
Concepto	**Importe (€)**
Consumo eléctrico de herramientas y equipos	45.00
Otros consumos eléctricos (calefacción, iluminación, etc.)	75.00
Total	**120.00**

Tabla 9.14.Total de costes indirectos.

9.3. Coste total del proyecto

El coste total del proyecto vendrá dado por la suma de todos los costes parciales, por lo que el presupuesto final vendrá dado por la suma de los costes directos e indirectos que se muestran en la tabla 9.15.

Coste total del proyecto	
Concepto	**Importe (€)**
Costes directos	19693.90
Costes indirectos	120.00
Total	**19813.90**

Tabla 9.5.Coste total del proyecto.

Capítulo 10:

Conclusiones y mejoras

En este último capítulo se enumerarán las conclusiones obtenidas del desarrollo de este proyecto, y además, se propondrán una serie de medidas que se podrían aplicar para mejorar la calidad del robot como plataforma robótica genérica y en el sistema de navegación.

La conclusión más clara es que se han conseguido alcanzar plenamente los objetivos principales planteados al inicio del proyecto, logrando además que el robot sea autónomo. Adquiriendo, de paso, un gran conocimiento sobre este tema. Adicionalmente, con esta memoria se facilita la documentación necesaria para que pueda continuarse el desarrollo de esta plataforma.

Se han adquirido nuevos conocimientos, tanto de la documentación previa al proyecto, de la que se obtuvo mucha información teórica, como durante el desarrollo del mismo, en donde se aplicaron todos los conocimientos adquiridos tanto en la etapa anterior como durante la carrera. De esta forma el proyecto ha sido muy útil también para el autor, que ha adquirido nociones sobre Robot Operating System y sobre Raspberry Pi, además de otros conocimientos variados de todas las ramas relacionadas.

En primer lugar se estudió a fondo el trabajo previo realizado y las soluciones adoptadas a la problemática planteada. Como ya se dijo anteriormente se había desarrollado una plataforma robótica que disponía de un ordenador portátil en la parte superior y que utilizaba la infraestructura de control Play/Stage para su manejo. A nivel inferior utilizaba una tarjeta GPMRC y varias placas controladoras y drivers de motores, además de los sensores y actuadores. La modificación de este nivel inferior no ha sido incluida en la realización de este proyecto debido a su correcto funcionamiento.

Sin embargo sí que ha habido que estudiar el funcionamiento de la GPMRC, ya que es el nexo de unión entre el hardware de la plataforma y la Raspberry Pi, al que se envían las órdenes y del que se reciben las lecturas de los distintos sensores y encoders.

Además del conjunto de sensores del que se partía se ha añadido un sistema láser para realizar una navegación mediante SLAM (Simultaneus Localization And Mapping).

En relación a la utilización de una plataforma embebida genérica, como es Raspberry Pi, como sistema de control a alto nivel, añade la ventaja de multiplicar las posibilidades de usos del robot. Gracias a ello se podrán desarrollar cuantas aplicaciones se necesiten en un futuro, sin tener que depender de compatibilidades con programas ya desarrollados, que además no suelen ser gratuitos. Además, como se explicó en el proyecto se dota al robot de más robustez, facilita las labores de mantenimiento y se abaratan los costes. Adicionalmente, el haber dotado a la Raspberry Pi de comunicación con un ordenador de sobremesa a través de Wi-Fi facilita su manejo.

Se ha podido valorar también la utilidad de dividir la aplicación de navegación en varios nodos, unidos y comunicados entre sí a través de una infraestructura de control como es ROS. Gracias a esta división se ha tenido una mayor capacidad de detección y localización de fallos, y además facilita enormemente la ampliación y reutilización de estos nodos para otros robots. Sin embargo este ha sido el apartado más conflictivo de todo el proyecto, surgiendo varios errores tanto en la instalación como en la gran labor de adaptación de los módulos del software al nuevo sistema.

Con el fin de implementar el sistema de control conjunto entre la Raspberry Pi y el ordenador, se ha utilizado una conexión mediante Wi-Fi, que además de permitir el trabajo conjunto, facilita mucho el manejo de la plataforma desde la mesa de trabajo.

Para terminar con el desarrollo del proyecto se ha sometido a la plataforma a una amplia batería de pruebas en la que se han encontrado problemas menores que se han ido subsanando de forma adecuada.

Además de los objetivos que se plantearon en el primer capítulo, la realización de este proyecto ha supuesto un trabajo donde aprender ha sido lo fundamental, aplicando conocimientos de la carrera y adquiriendo otros nuevos.

Finalmente hay que destacar que se ha conseguido implementar una aplicación plenamente viable para un robot móvil, que tiene muy pocos competidores de similares características en el mercado, obteniendo este resultado además a un bajo coste. Entre las prestaciones conseguidas se encuentra la posibilidad de navegación a través de SLAM o la integración del control en un sistema embebido, con lo que se ha conseguido incluir todos los elementos hardware dentro de la plataforma robótica.

A continuación se expondrán las mejoras, que al igual que en este caso, podrían plantearse para un posible nuevo trabajo o simplemente para perfeccionar su funcionalidad.

Como ya se explicó en apartados anteriores, cuando el robot está navegando por medio de un mapa guardado en la memoria, y detecta un nuevo obstáculo, se detiene, interrumpiendo así, la ejecución del recorrido. Este problema podría ser solventado intentando rodear el obstáculo mediante ultrasonidos o con el propio laser, o replanteando el recorrido y eligiendo una ruta alternativa que evite la zona conflictiva.

Otra posible mejora sería el mapeado automático. En este caso el mapeo se realiza de un modo manual. Esta operación podría automatizarse para que en caso necesario, comience a moverse por la zona y a crear el mapa, esquivando automáticamente los obstáculos que vaya encontrando por el camino. De esta manera se le podría dar unas coordenadas destino y sin tener ningún mapa almacenado ir creándolo y navegando por la zona hasta encontrarlo.

Un avance que haría al robot mucho más independiente es la carga automática. El robot ya cuenta con indicadores de nivel de batería. La mejora sería desarrollar un cargador al cual se pueda conectar el robot autónomamente y una modificación en el funcionamiento para enviar al robot al cargador en caso de tener un nivel de batería bajo.

Por último, podrían incluirse un giróscopo digital o un dispositivo de similar funcionalidad, que aportaría una gran mejora de la odometría, al hacerse el cálculo de los giros a través de este sensor y no mediante los codificadores incrementales, que pueden encadenar fallos si alguna de las ruedas desliza etc. De modo que la distancia recorrida la medirían los codificadores incrementales mientras que el giróscopo se encarga de determinar la orientación del robot.

Bibliografía:

Configuración de escritorio remoto para Raspberry Pi.
http://www.geekopasion.es/2013/03/09/configurando-tightvnc-para-que-se-ejecute-al-iniciar-raspberry-pi/

How to setup WIFI on your Raspberry Pi – Raspbian
http://thepihut.com/pages/how-to-setup-wifi-on-your-raspberry-pi-raspbian

Programas de control para el robot Turtlebot sobre ROS
http://robotica.unileon.es/mediawiki/index.php/Fernando-TFM-ROS02

Sistemas embebidos
http://www.idose.es/sistemas-embebidos

Introducción a los sistemas embebidos
http://es.slideshare.net/jkovima/introduccion-a-los-sistemas-embebidos-presentation

Sistemas empotrados, por José Luis Villarroel Salcedo.
http:// webdiis.unizar.es/~joseluis/SE.pdf

Desarrollo robótico. Robot Operating System.
http://www.slideshare.net/vicegd/desarrollo-robtico-robot-operating-system-ros?qid=9ed1b22e-294d-4d19-8d3f-5cad56ba345&v=default&b=&from_search=6

Tutoriales de Robot Operating System.
ftp://ftp.heanet.ie/disk1/sourceforge/r/ro/robotqbo/TUTORIALES%20ROS.pd

How to Choose the Right Platform: Raspberry Pi or BeagleBone Black?
http://makezine.com/magazine/how-to-choose-the-right-platform-raspberry-pi-or-beaglebone-black/

Helpful comparison of the Arduino Uno, Raspberry Pi and BeagleBone
http://www.adafruit.com/blog/2014/05/06/helpful-comparison-of-the-arduino-uno-raspberry-pi-and-beaglebone-black-beagleboneblack-txinstruments-beagleboardorg/

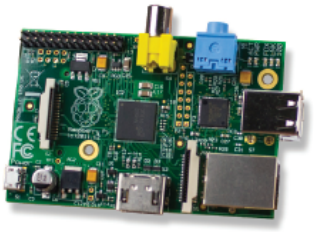
Cuatro alternativas a Arduino: BeagleBone, Raspberry Pi, Nanode y Waspmote
http://blogthinkbig.com/4-alternativas-arduino-beaglebone-raspberrypi-nanode-waspmote/

Comparativa entre Raspberry Pi, Arduino y BeagleBoard
http://developmentboards.blogspot.com.es/2013/04/raspberry-pi-vs-arduino-vs-beagleboard.html

Anexo A:

Datasheet Raspberry Pi.

Raspberry Pi



MODEL B

Product Name	**Raspberry Pi Model B**
Product Description	The Raspberry Pi is a small, powerful and lightweight ARM based computer which can do many of the things a desktop PC can do. The powerful graphics capabilities and HDMi video output make it ideal for multimedia applications such as media centres and narrowcasting solutions. The Raspberry Pi is based on a Broadcom BCM2835 chip. It does not feature a built-in hard disk or solid-state drive, instead relying on an SD card for booting and long-term storage.
RS Part Number	**756-8308**

Specifications

Chip	Broadcom BCM2835 SoC (a)
Core architecture	ARM11
CPU	700 MHz Low Power ARM1176JZFS Applications Processor
GPU	Dual Core VideoCore IV® Multimedia Co-Processor Provides Open GL ES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure
Memory	512MB SDRAM
Operating System	Boots from SD card, running a version of the Linux operating system
Dimensions	85.6 x 53.98 x 17mm
Power	Micro USB socket 5V, 1.2A (l)

Connectors:

Ethernet	10/100 BaseT Ethernet socket (b)
Video Output	HDMI (rev 1.3 & 1.4) (c); Composite RCA (PAL and NTSC) (d)
Audio Output	3.5mm jack (e), HDMI
USB 2.0	Dual USB Connector (f)
GPIO Connector	26-pin 2.54 mm (100 mil) expansion header: 2x13 strip. Providing 8 GPIO pins plus access to I²C, SPI and UART as well as +3.3 V, +5 V and GND supply lines (g)
Camera Connector	15-pin MIPI Camera Serial Interface (CSI-2) (h)
JTAG	Not populated (i)
Display Connector	Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane (j)
Memory Card Slot	SDIO (k)

Accessories

▲ Camera Module
775-7731

▲ International power supply
765-3311

▲ 8GB SD card pre-programmed with NOOBS - **779-6770**

▲ Expansion board
772-2974

▲ WiFi dongle
760-3621

▲ 10400mAh Li-Ion battery pack
775-7517

▲ Raspberry Pi user guide
768-6686

RS

www.rs-components.com/raspberrypi

Anexo B:

Datasheet láser Hokuyo.

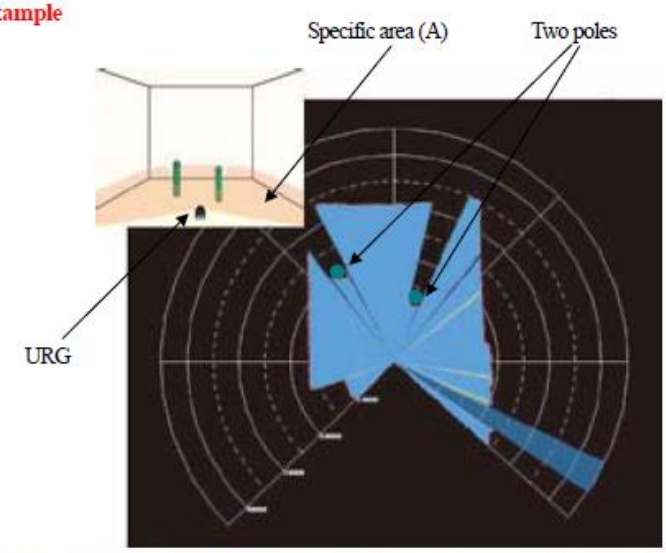
NEW PRODUCTS HOKUYO

SCANNING LASER RANGE FINDER FOR ROBOTICS **URG-04LX**

The Next Generation Area Detecting Sensor

Compact Design!
L:50, W:50, H70mm
Light Weight!
160g
Lower Power Consumption!
2.5W
High Accuracy!
±10mm
High Resolution!
0.36°
Wide Scanning Area
240°
Low Cost!

Scanning example



URG is scanning the specific area (A) and is detecting two poles and it shows distance and direction of two poles.

Applications

* Robot eyes, Security, Automatic doors, AGV(obstacle detecting sensor)

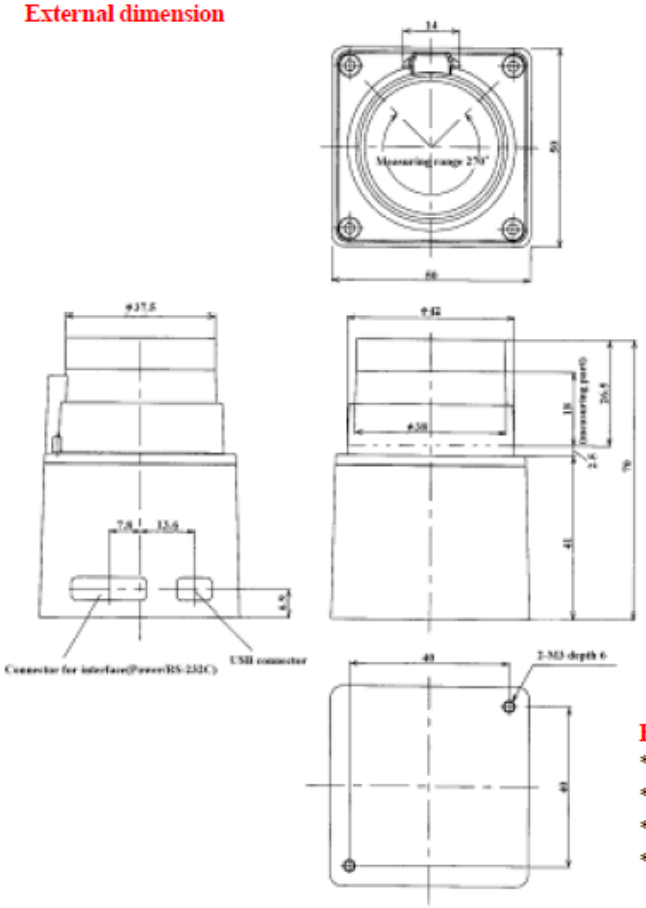
URG-04LX

HOKUYO

Specifications

Products	Scanning Laser Range Finder
Model No.	URG-04LX
Light source	Semiconductor laser λ=785nm, Laser safety class 1(IEC60825-1)
Power source	5VDC, ±5%
Current consumption	500mA or less(rush current 800mA)
Detectable distance and objects	20mm to 4000mm, white Kent sheet 70mm x 70mm*
Accuracy	(Official 20 to 1000mm : ±10mm, 1000 to 4000mm : ±1% of measurement) (White Kent sheet 70mm x 70mm)
Resolution	1mm
Scanning angle	240°
Angle resolution	Approx. 0.36° (360°/1024 steps)
Scanning time	100msec/scan
Interface	RS-232C(19.2k, 57.6k, 115.2k, 500k, 750kbps), USB : Ver.2.0 FS mode(12Mbps)
Ambient temperature/ humidity	-10 to +50°C, 85%RH or less(without dew and frost)
Protective structure	Optics : IP64, Case : IP40
Weight	Approx.160g
Material	Polycarbonate

External dimension



Interface

	URG-04LX	Cable colors
1	NC	
2	NC	
3	OUTPUT(Synchronous output)	Black
4	GND(9-pin, D-sub connector 5 pins)	
5	RxD(9-pin, D-sub connector 3 pins)	
6	TxD(9-pin, D-sub connector 2 pins)	
7	0V	Blue
8	5VDC	Brown

Exemption clause

* This sensor is not a safety instrument/tool.
* This sensor is designed for indoor use only.
* This sensor is not for use in military application
* Read this specifications sheet before using.

PHOTO SENSOR, LASER SENSOR, MICROWAVE SENSOR, COUNTER, AUTOMATIC DOOR

HOKUYO AUTOMATIC CO.,LTD.
1-10-9 Niitaka,Yodogawa-ku,Osaka
532-0033,Japan
Tel(06)6394-2102 Fax(06)6394-2339
http://www.hokuyo-aut.jp
E-mail:info@hokuyo-aut.jp

No.PR-12