ScienceDirect
Disponible en www.sciencedirect.com

Revista Iberoamericana de Automática e Informática industrial 11 (2014) 315–326 www.elsevier.es/RIAI

Desarrollo de un robot móvil compacto integrado en el middleware ROS

André Araújo[a], David Portugal[a,*], Micael S. Couceiro[a,b], Jorge Sales[c] y Rui P. Rocha[a]

[a] *Instituto de Sistemas e Robótica (ISR), Universidade de Coimbra, Portugal.*
[b] *RoboCorp, Departamento de Engenharia Electrotécnica, Instituto de Engenharia de Coimbra, Portugal.*
[c] *IRSLab, Departamento de Ingeniería y Ciencia de Computadores, Universitat Jaume I, Castellón, España.*

Resumen

En este trabajo se presenta el robot *TraxBot* y su integración completa en el *Robot Operating System* (ROS). El *TraxBot* es una plataforma de robótica móvil, desarrollada y ensamblada en el Instituto de Sistemas y Robótica (ISR) Coimbra. El objetivo de este trabajo es reducir drásticamente el tiempo de desarrollo, proporcionando abstracción de *hardware* y modos de operación intuitiva, permitiendo a los investigadores centrarse en sus motivaciones principales de investigación, por ejemplo, la búsqueda y rescate con múltiples robots o robótica de enjambres. Se describen las potencialidades del *TraxBot*, que combinado con un controlador de ROS específicamente desarrollado, facilita el uso de varias herramientas para el análisis de datos y la interacción entre múltiples robots, sensores y dispositivos de teleoperación. Para validar el sistema, se llevaron a cabo diversas pruebas experimentales utilizando robots reales y virtuales.

Palabras Clave:
ROS, robot móvil, sistemas embebidos, diseño, *middleware*, montaje y test.

1. Introducción

En este artículo se presenta el diseño e implementación del *TraxBot*, un robot móvil terrestre, desarrollado en el Laboratorio de Robótica Móvil (MRL), en el Instituto de Sistemas y Robótica (ISR) de la Universidad de Coimbra y su integración con el *middleware* ROS, a través de un controlador, beneficiándose así de la reutilización del *software* de su extensa comunidad de desarrolladores.

El *TraxBot* es ideal para la educación y la investigación, ya que puede proporcionar elementos básicos necesarios para el desarrollo de robots autónomos, tanto a nivel de *hardware* (mecánica, autonomía energética, locomoción, electrónica incorporada, sensores) como a nivel de *software* (teoría de control, programación de microcontroladores, planificación de trayectorias de navegación de robots, localización, etc.). La creación, desarrollo y programación del robot ha venido motivada por la experimentación y la investigación en sistemas de múltiples robots cooperativos, más específicamente, en equipos de robots con control distribuido para realizar patrullaje cooperativo (Portugal y Rocha, 2011) y forrajeo de enjambres (Couceiro *et al.*, 2011).

La robótica móvil es un área de investigación que ha sido testigo de increíbles avances de las últimas décadas. Temas como la navegación autónoma, la percepción, la reactividad, la creación de mapas, la evasión de obstáculos y la auto-localización, han sido profundamente estudiados durante los últimos años, con aplicaciones en la industria, el transporte, el área militar y entornos de seguridad (Petrina, 2011). Además, algunos robots móviles se han convertido en productos de consumo para el entretenimiento o para realizar las tareas domésticas diarias, como por ejemplo, aspirar el suelo (Jones *et al.*, 2006).

Anteriormente, el enfoque de la investigación se centró en estructuras grandes y medianas. Sin embargo, con los recientes avances en electrónica, la miniaturización de los sensores y la potencia de cálculo, el énfasis se ha puesto en el desarrollo de robots más pequeños y de menor coste, lo que hace posible la experimentación con grupos grandes de robots.

En el contexto de la investigación, se han desarrollado varios sistemas robóticos en distintos campos como la búsqueda y rescate, patrullaje, aplicaciones de seguridad, sistemas cooperativos de robots, interacción robótica humana o las competiciones de fútbol con robots y, hoy en día, casi todos los grandes institutos de ingeniería albergan uno o más laboratorios que se centran en la robótica móvil de investigación.

Con frecuencia, se siente la necesidad de herramientas prácticas de integración para implementar contribuciones científicas de valor. Sin embargo, en el área de investigación robótica, se pierde demasiado tiempo en soluciones de ingeniería para la configuración de *hardware* específico. Siguiendo la tendencia de la investigación, este trabajo se centra en una solución rentable y de código abierto, que permite a los investigadores en robótica, incluso a los estudiantes y entusiastas, la entrada en el mundo de la robótica y la inteligencia artificial.

Por tanto, en este trabajo se presentan en detalle las características del *TraxBot*: precio, capacidad de comunicación,

* Autor en correspondencia.
Correos electrónicos: aaraujo@isr.uc.pt (A. Araújo), davidbsp@isr.uc.pt (D. Portugal), micaelcouceiro@isr.uc.pt (M. Couceiro), salesj@uji.es (J. Sales), rprocha@isr.uc.pt (R. P. Rocha)
URL: http://mrl.isr.uc.pt/

http://dx.doi.org/10.1016/j.riai.2014.02.009

autonomía e integración de sus módulos, entre otros. Desafortunadamente, con el crecimiento exponencial de la robótica, se han encontrado algunas dificultades en términos de escritura del *software* compatible para distintos robots. Si variamos notablemente el *hardware*, la reutilización de código no es trivial. Frente a esta tendencia, ROS (Quigley *et al.*, 2009) proporciona las bibliotecas y herramientas para ayudar a los desarrolladores de *software* a crear aplicaciones robóticas, tal como se muestra en este artículo. Los objetivos principales de ROS son la abstracción de *hardware* de bajo nivel de control de dispositivos, la implementación de funcionalidades de uso común, el paso de mensajes entre procesos y la gestión de paquetes (*ROS packages*). Actualizaciones periódicas permiten a los usuarios obtener, construir, escribir y ejecutar código ROS en varios equipos y, por lo tanto, se muestra que es muy beneficiosa la integración de los robots en ROS, ya que reduce en gran medida el tiempo de desarrollo y permite un mayor enfoque en las tareas científicas. De este modo, las principales contribuciones de este trabajo son:

- Descripción del desarrollo y montaje del robot personalizado *TraxBot*, construido alrededor de una placa educativa Arduino.
- Desarrollo de un controlador de ROS para el *TraxBot*, lo que permite el control de la plataforma mediante ROS, la abstracción de *hardware* y reutilización de *software*.
- Evaluación de la plataforma y el controlador utilizando múltiples plataformas *TraxBot*.

En las dos secciones siguientes, se presenta una visión general del estado actual de la técnica, tanto de *hardware* y *software* para robots móviles como de robots móviles compactos de bajo coste. La sección 4 describe la plataforma *TraxBot* mientras que en la sección 5 se presenta el desarrollo del controlador de ROS y todas sus características. En la sección 6 se presentan los resultados del *driver* de ROS interactuando tanto con los robots virtuales simulados como con las plataformas *TraxBot* homólogas. Finalmente, la sección 7 resume las principales conclusiones y orientaciones para el trabajo futuro.

2. *Hardware* y *Software* para robots móviles

Todas las plataformas para robótica móvil, independientemente de su coste o propósito, comparten esencialmente la misma arquitectura básica de hardware. Podemos dividir el *hardware* en cuatro subsistemas: *i)* sistema de control; *ii)* sistema de accionamiento; *iii)* sistema de soporte, y *iv)* sistema de comunicación. El sistema de control comprende típicamente un microcontrolador y/o un ordenador incorporado, manejando todo el control de alto y bajo nivel necesario en la plataforma. El sistema de actuación, como su nombre indica, consiste en actuadores que, en la mayoría de plataformas se componen principalmente de los motores eléctricos, pero hay algunas excepciones, como los servos neumáticos. Complementariamente a la "decisión" que proporciona el sistema de control, el sistema de accionamiento define el "hacer" del proceso cognitivo del robot móvil. El sistema de soporte define la alimentación de las plataformas que, por otro lado, depende en gran medida el sistema de accionamiento (por ejemplo, baterías para motores eléctricos y compresor de aire para los servos neumáticos). Por último, el sistema de comunicación define la capacidad que el robot móvil tiene para compartir información (por ejemplo, la tecnología *WiFi*). Aunque esto no es imprescindible en muchas soluciones en las que sólo es necesario tener un único robot para fines de investigación en laboratorio, la mayoría de aplicaciones reales requieren que los robots puedan comunicarse, ya sea implícita o explícitamente, con otros robots u operadores humanos.

Independientemente de la arquitectura *hardware*, los robots móviles requieren un *middleware* que integre todos los componentes descritos anteriormente de una manera eficiente. Un *middleware* utilizado en robots móviles autónomos es típicamente embebido, concurrente, en tiempo real, distribuido y debe garantizar propiedades del sistema tales como la seguridad, la confiabilidad y la tolerancia a fallos. Hoy en día, existe una amplia gama de opciones de *middleware* para la robótica, que van desde los enfoques orientados a la industria, tales como *OROCOS* (OROCOS, 2014), hasta los enfoques orientados a robots humanoides, como *YARP* (YARP, 2014).

En (Mohamed *et al.*, 2008), los autores revisan algunos de los *middleware* más conocidos hasta el año 2008 y se discuten los desafíos y ciertos problemas por resolver inherentes a la robótica. En lugar de proporcionar la mejor solución global de *middleware*, clasifican cada *middleware* en cuanto a su objeto principal. Entre la lista, que es de tamaño considerable, se presta especial atención a *ORCA* (ORCA, 2014), que además de ser definido como modular, con abstracción de alto nivel, y un paradigma basado en componentes, presenta un alto nivel de reutilización del código. La reutilización es una capacidad que también es compartida por el *UPnP Robot Middleware* (Ahn *et al.*, 2006), que incluye, además, funcionalidades de tiempo real, y MARIE (MARIE, 2014), que se define por su alto nivel de flexibilidad y expansión de funcionalidades. Las características del conocido *framework Player/Stage* (Player/Stage, 2014) también son discutidas por los autores, destacando sobre todo la facilidad para mejorar funcionalidades e incorporar otras nuevas en sistemas de robots existentes.

Dando especial relevancia a la reusabilidad del código, Brugali y Scandurra (Brugali y Scandurra, 2009) propusieron un *framework* de ingeniería *model-driven* en el contexto del proyecto europeo de buenas prácticas en robótica (BRICS, 2014). A pesar de los diferentes requisitos y cuestiones abiertas, los autores destacan la relevancia de la reusabilidad del código, indicando la necesidad de mantener el ritmo de la investigación robótica y no tener que escribir código ya existente.

La relevancia de reusabilidad del código se consideró como un elemento clave en el diseño de recientes *frameworks* robóticos. En (Schlegel *et al., 2011*), los autores comparan la solución propuesta en (Brugali y Scandurra, 2009), ROS (Quigley *et al.*, 2009) y su método SMARTsoft (SMARTsoft, 2014). Los autores abordan las ventajas de un método *model-driven* sobre un método *code-driven*, como es el caso de ROS. Sin embargo, según lo declarado por los propios autores, ROS es el *framework* más ampliamente utilizado, siendo por tanto representativo de la situación actual en términos de *software* de robótica, no existiendo actualmente ningún otra que asegure reusabilidad del código al mismo nivel.

De hecho, este ha sido uno de los objetivos principales detrás de ROS. ROS ya es el *middleware* más cerca de convertirse en el estándar que la comunidad robótica necesitaba con urgencia. Muchos *frameworks* de robótica en el pasado se han diseñado en respuesta a las debilidades de otros *frameworks* disponibles o para poner énfasis en los aspectos que se consideraban más importantes en el proceso de diseño. ROS es el producto de soluciones de compromiso y priorizaciones efectuadas durante este proceso.

ROS promueve la reutilización de código con diferente *hardware*, proporcionando una gran cantidad de bibliotecas disponibles para la comunidad, como SLAM 2D basado en laser (Machado Santos et al., 2013), SLAM visual (Grisetti *et al.*, 2007), y Nubes de Puntos basadas en el reconocimiento de objetos 3D (Rusu *et al.*, 2011), entre otros, así como herramientas para la visualización en 3D (rviz), grabación y reproducción *offline* de los experimentos de datos (rosbag), y mucho más.

ROS permite reutilizar código de numerosos otros proyectos de código abierto, tales como varios *drivers* de *Player*, los simuladores Stage 2D (Gerkey *et al.*, 2008) y Gazebo 3D (Gazebo, 2014), el *stack* Orocos, que integra este *middleware* más utilizado en robots industriales y control de máquinas, los algoritmos de visión de OpenCV (OpenCV, 2014) y funcionalidad parcial del Mobile Robot Programming Toolkit (MRPT, 2014). Más allá de la simplicidad de uso de las herramientas y algoritmos disponibles, ROS también proporciona una fácil integración de nuevos sensores, sin necesidad de conocimientos de *hardware*. Como resultado, el tiempo total invertido en el desarrollo de *software* se reduce en gran medida debido a la reutilización de código y de abstracción de *hardware*, y la integración de los robots y los sensores en ROS resulta beneficiosa.

3. Robots móviles compactos y de bajo coste

En esta sección se revisan varios robots populares de bajo coste con sensores simples y capacidades diversas, teniendo en cuenta la movilidad del robot en diferentes ambientes de tierra, capacidades, tamaño (hasta 35 cm de diámetro o diagonal), capacidad sensorial y de percepción, potencia de procesamiento, navegación autónoma y otros aspectos del diseño. Todos los robots de bajo coste mencionados están disponibles en el mercado o pueden ser fácilmente construidos por un precio de hasta 450€, permitiendo tanto a investigadores, como a los consumidores en general, desarrollar soluciones diversas con sensores y capacidades simples.

El robot Create de iRobot (Kuipers, 2009) es muy popular en la comunidad robótica debido a su pequeño tamaño y bajo coste. Consiste en una plataforma circular con aproximadamente 340 mm de diámetro, que soporta equipos de hasta 2,26kg con un espacio extra para sensores más grandes. Muchos eligen utilizar un ordenador externo que soporta comunicación en serie para controlar el robot, debido a las limitaciones en cuanto a espacio de almacenamiento y potencia de procesamiento. De hecho, un controlador de ROS para el Create ya ha sido desarrollado (Roomba, 2014).

Otro robot de bajo coste es el Bot´n Roll ONE C de SAR (SAR, 2010). Este robot tiene una configuración diferencial, proporcionando dos sensores infrarrojos de detección de obstáculos con la posibilidad de añadir módulos adicionales. Un conversor USB-Serie (RS232) permite la programación del robot mediante un ordenador externo. En general, se trata de un kit de iniciación excelente para los principiantes.

El kit robot Circular GT de IdMind (IdMind, 2005), con una forma circular de 150 mm de diámetro, apoyada en una placa de plástico y ruedas diferenciales, es más pequeño que el Bot'n Roll ONE C, y tiene más puertos E/S para conectar posibles extensiones que uno mismo pueda diseñar. Posee cinco pares de sensores de infrarrojos, dos micro-interruptores para la detección de colisiones y 7 enlaces adicionales para conectar otros sensores.

El Hemisson de K-Team (Colot, 2006) es más robusto que el IdMind´s Circular GT. El kit básico proporciona una potencia computacional limitada y unos pocos sensores, como 8 sensores de luz infrarroja, 6 de ellos para detección de obstáculos y 2 orientados hacia el suelo. Permite la programación en memoria flash y es compatible con cámaras y sensores (*e.g.*, ultrasónicos). Estas extensiones permiten incrementar tanto la potencia de cálculo como la capacidad de detección.

El Mindstorms NXT de Lego es un kit robótico educativo y académico para principiantes (Bagnall, 2007). El robot está equipado con motores y codificadores ópticos (*i.e.*, *encoders*) y una buena variedad de sensores, como un acelerómetro, sensores de luz, sonido, ultrasonido y táctiles que permiten aplicaciones en una amplia gama de escenarios. También soporta comunicación Bluetooth y USB. Un soporte muy limitado para el control de este robot con ROS está ya disponible.

El SRV-1 Blackfin de Surveyor (Cummins *et al.*, 2008), es un robot equipado con cuatro motores y orugas de estilo tanque con tracción diferencial con una longitud de 120 mm, una anchura de 100 mm y un chasis de aluminio. Este robot tiene una capacidad de procesamiento considerable, con una CPU que ofrece 1000 MIPS a 500 MHz, capaz de ejecutar un Linux con Kernel 2.6. Está equipado con dos láseres de rango o de manera opcional, con sensores ultrasónicos (admite hasta 4 módulos de ultrasonidos) y una cámara de 1.3MP. También soporta red inalámbrica 802.11b/g y varios sensores I2C.

Tabla 1: Cuadro comparativo de los robots móviles compactos de bajo coste.

	Roomba iCreate	K - Team Hemisson	Idmind Circular GT	Bot'n Roll ONE C	Lego Mindstorm NXT	Surveyor SRV-1 Blackfin
						
Precio Base Aprox.	**180,00 €***	**275,00 €**	**260,00 €**	**175,00 €**	**380,00 €**	**275,00 €**
Especificaciones físicas :						
- Dimensiones (LxAxA) [mm]	(-/-/90)	(-/-/70)	(-/-/60)	220x205x90	145x97x61	120x100x80
- Diámetro [mm]	Ø 340	Ø 120	Ø 150	-	-	-
- Peso [g]	3400	200	200	1300	350	200
- Material del chasis	Plástico	Aluminio	Plástico	Acrílico	Plástico	Aluminio
Hardware:	**					
- Procesador	20MIPS 20MHz	5MIPS 20MHz	5MIPS 20MHz	16MIPS 32MHz	30MIPS 40MHz	1000MIPS 500MHz
- µControlador	ATmega 168	PIC16F877	PIC16F877	Picaxe 40x2	ARM 7	-
Sensores:						
	- Temperatura - Detector de obstáculos - µInterruptores para la detección del desniveles	- 2 sensores de luz (iR) - 6 detectores de obstáculos (iR)	- 7 detectores de obstáculos (iR) - 2x µinterruptores	- 2 detectores de obstáculos (iR)	- Acelerómetro - Sensor táctil - Sensor de luz - Sensor de sonido - Sensor ultrasónico	- Cámara 1.3 MP - 2 Lasers rango
Comunicación:						
	- RS-232 serial - USB serie	- DB9 Serie	-	- USB serie	- Bluetooth - USB	- Wireless 802.11b/g
Velocidad:						
Máx (cm/s)	-	10	-	-	10	40
Batería :						
- Tipo	Ni-MH	Ni-MH	Alcalina AA	Ni - MH	Li-Ion AA	Li-poli
- Potencia	14.8V 3000mAh	8.4V 150mAh	6V= 4x 1.5V	12V	9V = 6x1.5V	7.4V 2000mAh
- Autonomía aprox.(horas)	3 - 4	2	2	2	3 - 4	3
Características:						
	* Por lo general, necesita microcontrolador externo. ** con módulo de control.	- Soporta otros sensores y módulos de comunicación - Zumbador	-	- Xbee (conexión) - LCD (conexión) - Soporta otros sensores - Zumbador	- Pantalla LCD - Diferentes configuraciones con piezas de Lego	- Linux 2.6 - Puntero laser - Consola basada en Java

3.1. Discusión

En la Tabla 1 se presenta una comparación de todos los robots móviles compactos de bajo coste descritos anteriormente. Se puede observar que el robot iRobot Create es el más barato de todos. Sin embargo, es muy pobre en capacidad de detección, resolución del codificador óptico (*i.e.*, *encoder*) y potencia de procesamiento, por lo que en general necesita un ordenador portátil externo instalado en la parte superior. Por otra parte, el SRV-1 Blackfin, es uno de los más caros, pero proporciona la mejor capacidad de procesamiento y es el único que viene con una cámara. No obstante, la posibilidad de añadir extensiones en forma de sensores es limitada, debido a la falta de espacio disponible. Lo mismo ocurre con Lego Mindstorms NXT, que

tiene menos capacidad de procesamiento. El Circular GT de IdMind tiene un coste muy efectivo, pero no dispone de *hardware* para permitir la comunicación entre múltiples robots, por lo que es menos adecuado para la experimentación con equipos de robots. Lo mismo puede decirse del Bot'n Roll One C. En cuanto el K-Team Hemmisson, su desventaja es la autonomía.

Hay que tener en cuenta que, en su forma actual, la mayoría de estos robots están limitados a tareas muy sencillas, ya que la odometría es poco fiable. Los robots equipados con sensores son más adecuados para tareas que requieren una localización más precisa. Por lo tanto, en la mayoría de estas plataformas se necesitan extensiones para que sus capacidades de detección de tareas sean más exigentes.

Además del sistema sensorial limitado, ninguno de estos robots es completamente capaz de ejecutar ROS, u otro software de robótica, por sí mismo. Necesitarían de una CPU externa adicional para ejecutar una versión de Linux compatible, *i.e.*, Ubuntu, Debian o OS X o, en el caso del SRV-1 Blackfin, instalar ROS a partir del código fuente, realizando una compilación sobre la arquitectura del robot, algo que nunca es trivial.

Las muchas limitaciones de los robots una vez revisado el estado de la técnica en esta materia, ha motivado el desarrollo e integración de la plataforma *TraxBot* (Araújo *et al.*, 2012) en el *middleware* ROS.

4. La Plataforma TraxBot

El diseño del *TraxBot*, que se describe más adelante, ha sido elegido atendiendo esencialmente a las siguientes razones:

- **Robustez:** Todo el hardware es de aluminio o acero inoxidable;
- **Coste:** El coste de la plataforma es de alrededor de 470€;
- **Operabilidad:** Tiene la capacidad de maniobrar en muchos tipos diferentes de terreno y topografías de superficie;
- **Autonomía energética:** Puede funcionar continuamente alrededor de 2-3 horas;
- **Sistema de sensores:** Equipado con sensores ultrasónicos de rango para permitir la interacción con el medio ambiente;
- **Dimensiones:** Es adecuado para experimentos tanto en interiores como en exteriores;
- **Flexibilidad:** Se le pueden incorporar muchas extensiones nuevas y componentes (*e.g.*, LED's, cámaras, LIDARs, pinzas, etc.);
- **Diseño híbrido:** Es capaz de trabajar con y sin un pequeño *netbook* en la parte superior de la plataforma de acuerdo con los requerimientos computacionales del usuario;
- **Comunicación:** Soporta comunicación inalámbrica ZigBee o 802.11b/g cuando se utiliza el *netbook*;
- **ROS-compatible:** Capacidad para utilizar herramientas de ROS.

Teniendo en cuenta la falta de flexibilidad y las debilidades de los robots móviles compactos descritos anteriormente, sobre todo en términos de capacidades de comunicación, de detección y de procesamiento, nuestro grupo de investigación ha desarrollado este robot con herramientas económicas y de código abierto. Al describir el *TraxBot* en detalle en las secciones siguientes, se muestra que es una plataforma ideal para fines educativos e investigación y, aunque es un poco más caro que los robots revisados, los autores creen que es mucho más rentable, dado el conjunto de sus capacidades.

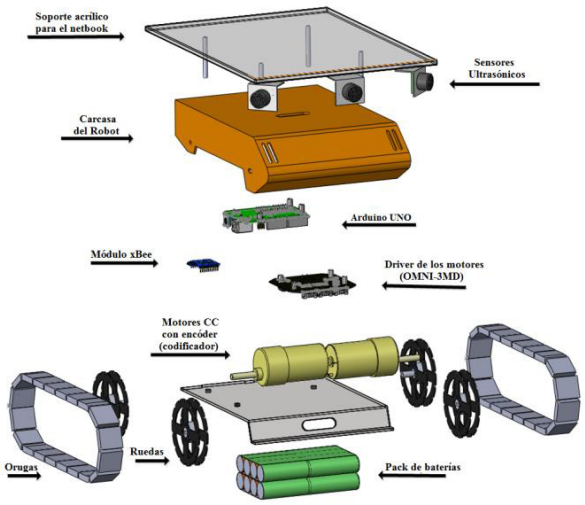

Figura 1: Estructura mecánica del *TraxBot*.

4.1. Descripción General del Hardware

El *TraxBot* es una plataforma robótica móvil tipo oruga, que tiene un sistema de accionamiento diferencial construido sobre el kit comercial Traxster II. El ligero y robusto chasis de aluminio está equipado con dos motores de corriente continua (CC) con reductora en las ruedas delanteras, con codificadores ópticos de rueda en cuadratura con una resolución de 624 pulsos. Los motores funcionan a 7,2 voltios, siendo capaces de alcanzar 160rpm con 7,2 kg-cm de par sin carga. Dado que la plataforma original está dotada de orugas modeladas con plástico ABS inyectado, se han añadido bandas adicionales de goma para aumentar la fricción y reducir el deslizamiento. Se añadió también una estructura plana acrílica en la parte superior de la plataforma para poder soportar un *netbook* de 10" y los tres sensores ultrasónicos (Fig. 1).

Las unidades de procesamiento y control están formadas por un Arduino Uno con un *shield* XBee y una unidad de motor OMNI-3MD (OMNI-3MD, 2011), que se alojan en el interior del chasis del *TraxBot*. Con el objetivo de garantizar una buena autonomía y la potencia requerida, dispone de dos paquetes de baterías de 12V y 2300mAh de Ni-MH, que constan de 8 celdas, cada una, de Ni-MH tipo AA. Estas baterías están colocadas bajo el *TraxBot* y vienen fijadas por dos tiras de velcro en la parte exterior, proporcionando un fácil acceso y reemplazo. El interruptor de alimentación del circuito se encuentra en la parte posterior del chasis. La Tabla 2 presenta algunas especificaciones de *hardware*.

Tabla 2: Especificaciones de *Hardware* del *TraxBot*.

Especificación	Valor	Unidades
Rango de Voltaje	9-14	V
Corriente eléctrica en operación	1200	mA
Corriente eléctrica en reposo	110	mA
Velocidad máxima	0,95	m/s
Peso	2045	g
Ancho	203	mm
Longitud	229	mm
Altura	125	Mm
Autonomía Energética	2-3	horas

4.2. Arquitectura y Componentes

TraxBot se presenta como una plataforma ideal para robótica educativa, ya que se basa en la placa Arduino Uno que es a su vez una plataforma de código abierto y presenta un interfaz fácil de utilizar para la comunicación, permitiendo el intercambio de datos entre múltiples robots o entre el robot y una unidad remota.

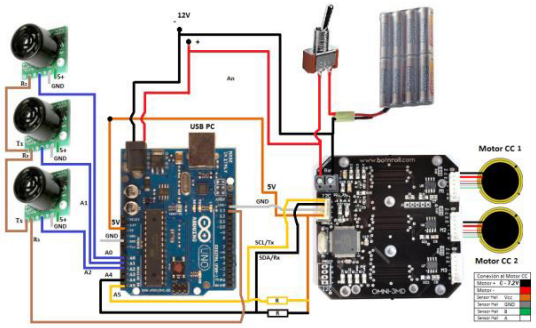

Figura 2: Esquema eléctrico del *TraxBot*. De izquierda a derecha: Tres sónares, placa Arduino Uno, interruptor de alimentación, baterías, *driver* de los motores OMNI-3MD y dos motores CC.

En esta sección se describe la organización de los componentes del interior de la plataforma y la arquitectura de control. En la Fig. 2 se presenta un esquema de la electrónica de la plataforma *TraxBot*. La placa Arduino Uno, el *driver* Bot'n Roll OMNI-3MD y los motores de CC se encuentran dentro del chasis del robot, mientras que los sónares Maxbotix MB1300 se alojan bajo la plataforma acrílica, proporcionando un diseño configurable. Tener el Arduino Uno como el componente central del sistema es útil para conectarse a través de los pines analógicos A0, A1 y A2. En cuanto a la conexión con el *driver* OMNI-3MD de los motores, los pines A4 y A5 se utilizan para la conexión I2C. Los canales de los codificadores ópticos de cada motor se conectan a la placa OMNI-3MD. El conector USB del microcontrolador se conecta al *netbook*, que se utiliza para recibir (RX) y transmitir (TX) datos serie TTL decodificados usando un chip *USB-a-Serie*.

4.2.1 Procesamiento y control

La unidad de procesamiento consiste en un microcontrolador (μC) de 8 bits ATmega328p de Atmel, incrustado en la placa *open-hardware* Arduino Uno (Arduino, 2010). Un único circuito integrado contiene los 3 componentes principales de la arquitectura: unidad central de procesamiento (CPU), memoria y entrada/salida (E/S).

El Arduino tiene un entorno de desarrollo multiplataforma y de código abierto con características extensibles de *software* y *hardware*, es decir, el código se puede ampliar a través de bibliotecas de C (como se verá en la sección 5). Además se pueden conectar *shields* en la parte superior de la placa Arduino, extendiendo así sus capacidades. En nuestro caso, se ha añadido un módulo *shield* ZigBee (véase la sección comunicación 4.4). El Atmega 328p, como muchos otros μCs, puede ofrecer una flexibilidad limitada debido a sus inherentes limitaciones de hardware, en comparación con otros sistemas de procesamiento (*e.g.*, PCs integrados). Aún así son la mejor opción cuando se crean dispositivos electrónicos simples que no cambiarán mucho con el tiempo (*e.g.*, robótica de enjambre). Con el fin de superar estas limitaciones, en caso necesario, se utiliza un *netbook* de 10", para procesamiento externo, que se dedica a ejecutar ROS. En particular, se utiliza un ASUS eeePC 1001PXD BLACKN455, con un procesador Intel Atom a 1,66 GHz. Esta es una solución económica, con un tamaño adecuado para este robot y capaz de ejecutar ROS con todas las funcionalidades, ofreciendo también comunicación Wi-Fi. Este equipo proporciona un diseño híbrido que permite al robot realizar una gran variedad de tareas. Si se necesita más potencia de cálculo, es posible utilizar múltiples máquinas (CPUs), para reducir la carga de cálculo de un único equipo (como se verá más adelante en la sección 6.4). En este trabajo, el μC ATmega 328p envía comandos de velocidad lineal y angular al *driver* de los motores OMNI-3MD. Este *driver* tiene la capacidad de controlar hasta tres motores en plataformas omnidireccionales, recibiendo comandos de velocidad y traduciéndolos en comandos de accionamiento para los motores. Así, se puede realizar el control tanto de velocidad como de posición. Los motores funcionan con una tensión de entre 7 y 24 Vcc.

Dirección	L/E	INSTRUCCIÓN	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	...	BYTEn
0x18	E=0		Bytes de Datos						

Figura 3: Trama del protocolo OMNI-3MD a través del bus I2C.

En cuanto a la monitorización, es posible acceder a la tensión de cada motor, a la versión de *firmware*, así como a la temperatura de la placa. Por otra parte, la placa OMNI-3MD tiene la flexibilidad de proporcionar directamente el número de pulsos de los *encoders*, que se pueden configurar un *prescaler* deseado. Para el control de los motores, la placa proporciona un PID de bucle cerrado.

Para la conexión de datos en serie y la conexión del reloj en serie, el BUS I2C trabaja con una tasa de intercambio de datos de hasta 100 kbit/s en el modo estándar o hasta 400 kbit/s en modo rápido. Para el intercambio de datos I2C, se utiliza una trama con 3 segmentos para la dirección, el comando y los datos (Fig. 3). En primer lugar, el segmento de 8 bits contiene la dirección I2C, donde el bit más significativo selecciona el modo de escritura o de lectura. El siguiente segmento es la lista específica de los comandos de operaciones, y el último segmento de la trama son los datos de los parámetros de los comandos. Para su protección, si la conexión I2C excede un tiempo de espera mayor, el *driver de los motores* interrumpe automáticamente los movimientos de los motores. Después de haber especificado el *hardware* y la electrónica de la plataforma, en la Fig. 4 se resume la arquitectura de control modular del robot.

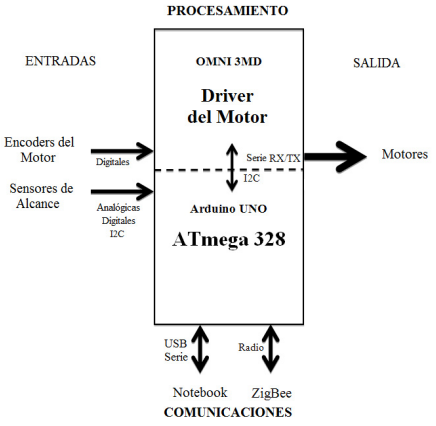

Figura 4: Arquitectura de control de la plataforma robótica.

4.3. Sónares y Odometría

Para la detección de rango, el *TraxBot* utiliza 3 sónares ultrasónicos de rango, más precisamente, 3 LVMaxSonarMB1300 de Maxbotix (MB1300, 2005), con una zona muerta de 15 cm, un rango máximo de aproximadamente 6,45m, resolución de 2,45 cm y un diseño configurable. Es posible utilizar hasta 4 sónares en una plataforma a través de los puertos analógicos disponibles en la placa Arduino UNO.

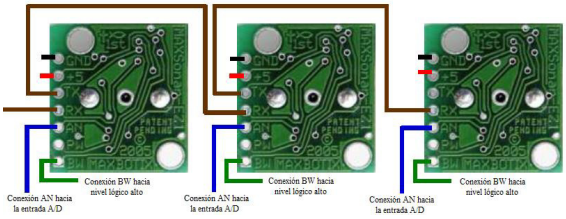

Figura 5: Conexión de la cadena de sónares.

Estos sónares ultrasónicos envían un pulso de sonido y esperan a recibir el eco del sonido rebotado desde un obstáculo. Midiendo el tiempo que tarda el sonido en estar de vuelta, el µC embebido en la propia placa del sónar calcula la distancia que el sonido ha recorrido, y a partir de ahí, cuán lejos está el objeto. Sin embargo, esto origina problemas cuando diferentes sónares "escuchan" un eco de otros sónares, que se traducen en detecciones erróneas. De ahí que, para evitar este fenómeno de *crosstalk*, se utiliza un bucle en cadena para tener valores de cadencia sincronizada de los sónares individuales, tal como se muestra en la Fig. 5.

Los dos motores de CC utilizados en este proyecto tienen un codificador óptico de cuadratura incorporado, con 624 impulsos por revolución del eje de salida, que se utiliza para determinar la posición, así como la dirección y velocidad de rotación de la rueda mediante la conversión de desplazamiento en impulsos digitales (Data Translation, 2006).

4.4. Comunicaciones

El *TraxBot* proporciona comunicación inalámbrica, lo cual es fundamental para equipos de robots. La comunicación ZigBee está integrada en el *TraxBot* y el Wi-Fi 802.11 b/g/n también está disponible cuando se utiliza un ordenador portátil *(netbook)* en la parte superior del mismo. El estándar ZigBee se utiliza para intercambiar mensajes cortos entre robots cuando se opera sin un portátil *netbook* y se ejecutan algoritmos simples de coordinación (Portugal y Rocha, 2012), mientras que el Wi-Fi es compatible con mayor ancho de banda, para el intercambio de grandes datos y se integra fácilmente en una red con ROS (ver sección 5).

La Tecnología ZigBee proporciona una comunicación inalámbrica rápida y de bajo coste, con un bajo consumo de energía, y con la posibilidad de soportar un gran número de nodos (en teoría hasta 65536). Asimismo, permite establecer comunicaciones punto-a-punto, entre pares *(peer-to-peer)* y *multicast,* adecuadas para la comunicación de equipos de robots cooperativos. En este trabajo, el *TraxBot* se ha equipado con una *shield* Xbee (XBee, 2006) de MaxStream, que consiste en un módulo de comunicación ZigBee con una antena conectada en la parte superior de la placa Arduino Uno a modo de módulo de expansión. Esta *shield* opera el protocolo ZigBee estándar IEEE 802.15.4 utilizando la banda ISM de 2.4 GHz. Este módulo Xbee tiene una potencia de 1mW y alcanza entre 30 y 100 metros, para el uso en interiores y al aire libre, respectivamente. El consumo de energía de este módulo es extremadamente bajo: 10µA en modo de reposo y 50mA durante el envío y recepción de datos. El *netbook* proporciona comunicación inalámbrica al robot a través de Wi-Fi 802.11 b/g/n y se encarga de la ejecución de ROS, como ya se ha mencionado con anterioridad. La conectividad Wi-Fi se utiliza para el acceso remoto y para posibilitar la ejecución de ROS y poder compartir información.

4.5. Cinemática

A nivel cinemático, el *TraxBot* se considera un robot diferencial no holonómico (Fig. 6), con dos motores controlables de manera independiente. La configuración del sistema de locomoción basado en cadenas le proporciona un mejor agarre en la tracción y estabilidad, lo que permite realizar una gama más amplia de tareas en distintas superficies. Sin embargo, el sistema de locomoción utilizado presenta un efecto de deslizamiento que necesita ser resuelto. Por lo tanto, con el fin de controlar el *TraxBot* más eficazmente, el efecto del deslizamiento se minimizó con un *offset* de deslizamiento. Es necesario definir este *offset* de deslizamiento tanto en el caso de la rotación (1) como en el caso del desplazamiento lineal. Estos *offsets* (Cs) y (Cm) se determinaron en una alfombra suave, para el caso de una rotación adecuada y estable. Teniendo en cuenta la notación de la Tabla 3 y teniendo en cuenta el radio de la rueda R_w=39 mm y el radio de robot R_r=102 mm; la combinación entre *encoders* y la rueda proporciona NP = 624 pulsos/revolución, donde cada impulso corresponde a una resolución de aproximadamente 0.004 grados.

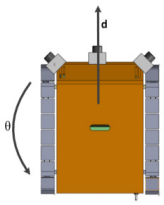

Figura 6: Cinemática del *TraxBot*.

Rotación:

$$D_r = -D_l = (N_p + C_s)\frac{\theta}{2\pi}\frac{R_r}{R_w} \quad (1)$$

Movimiento Recto:

$$D_r = D_l + C_m = N_p \frac{d}{2\pi R_w} + C_m \quad (2)$$

Tabla 3: Notación.

Descripción	Símbolo
Radio de las ruedas	R_w
Radio del robot	R_r
Número de pulsos enviados por el codificador óptico de la rueda izquierda	D_l
Número de pulsos enviados por el codificador óptico de la rueda derecha	D_r
Distancia entre la posición actual y la de destino	d
Ángulo de rotación entre la posición actual y la de destino	θ
offset de deslizamiento	C_s
Número total de pulsos por revolución leídos por los codificadores ópticos	N_p
Valor de compensación de los motores	C_m

Dada la información anterior, es posible obtener una resolución de 0,39 mm para cada pulso proveniente de los *encoders*.

4.6. El coste del TraxBot

La plataforma completamente ensamblada tiene un coste final de 470€, tal como se detalla en la Tabla 4.

Teniendo en cuenta todas sus capacidades, el *TraxBot* tiene una buena relación calidad-precio en comparación con los otros robots compactos descritos anteriormente. También proporciona la flexibilidad y el espacio necesario para incorporar muchas más expansiones, que van desde simples sensores basados en LED, micrófonos, pantallas; así como otros más complejos como sensores láser de rango (LRF) o un sensor Kinect, entre otros; opciones que no son comunes en otros robots del mismo tipo.

A pesar de que el *driver* del motor es una de las partes más costosas del robot, debe tenerse en cuenta que esta es una *board* muy potente tal como se describe en (OMNI-3MD, 2011). También es flexible ya que soporta tres motores, lo que aumenta las propiedades de expansión de la plataforma a fin de incluir un actuador adicional, por ejemplo, para tareas de manipulación. Además, el desarrollo de la *board* contó con la participación de los autores. Como consecuencia, hubo beneficios mutuos institucionales (entre una empresa privada y un instituto de investigación estatal en robótica), siguiendo una estrategia de *technology push*.

Al utilizar un *netbook* para beneficiarse de todo el potencial de ROS, el precio total de la plataforma se eleva ligeramente, *e.g.*, el Asus eeePC 1001PX cuesta alrededor de 180€. Sin embargo, esto también se aplica a los otros robots revisados en la sección 3, excepto teóricamente para el SRV-1 Blackfin (a pesar de que, hasta la fecha, no se conoce ninguna instalación con éxito de ROS a bordo del robot). No obstante, muchos de ellos (*e.g.*, Lego NXT) ni siquiera disponen del espacio necesario para incluir un *netbook*.

Tabla 4: Coste total en euros del *TraxBot*.

Descripción	Cant.	Coste [€]
Arduino UNO	1	13,00
Arduino XBee *shield* + módulo Xbee	1	24,00
Sensor ultrasónico de rango MaxSonar	3	72,00
Traxster II Robot Chassis Kit	1	151,00
Baterías 12V 2300mAh Ni-MH	2	36,00
Driver de los motores OMNI-3MD	1	149,00
Componentes electrónicos varios	-	25,00
	TOTAL	**470,00**

5. Desarrollo de los controladores ROS para el TraxBot

Una vez ensamblado el robot, es necesario un controlador *software*. Un controlador *software* permite la interfaz y el control de la plataforma y se comunica con el *hardware* de manera que los comandos enviados por el programa de alto nivel son interpretados correctamente por el *hardware* del robot. En este caso, el controlador interpretará programas de alto nivel en ROS.

Con este propósito se ha desarrollado un controlador para la plataforma (TraxBot Driver, 2014), compatible con ROS *Fuerte*. Para permitir la comunicación serie punto a punto, se utiliza el *stack rosserial* (Rosserial, 2014), haciendo uso de un protocolo general para intercambiar mensajes de ROS con el Arduino Uno. El *stack rosserial* es ampliamente utilizado por la comunidad de ROS, por ser de código abierto y estar en constante mejora. Este *stack* dispone de bibliotecas de cliente específicas que se pueden utilizar en el código fuente principal del microcontrolador. Estas bibliotecas permiten a los usuarios iniciar fácilmente nodos ROS mediante el servidor ROS instalado en la CPU del *netbook*. El nodo *rosserial* se utiliza para configurar el protocolo de conexión en el controlador del *TraxBot*. El *driver* de potencia del motor OMNI-3MD proporciona bibliotecas para controlar los motores (es decir, la velocidad o el control de posición), leer los *encoders* y la temperatura, así como para el establecimiento de los parámetros en la configuración inicial del controlador PID y el *prescaler* de los *encoders*, entre otros. El OMNI-3MD se conecta al Arduino Uno mediante comunicación I2C.

El código desarrollado para controlar el *firmware* del robot (Algoritmo 1) tiene en cuenta todos los componentes (y sus características) que son necesarios para la operación del robot. El resto de la arquitectura del controlador ROS desarrollado se representa en la Fig. 7. En el desarrollo del controlador, además del *stack rosserial* hacemos uso solamente de mensajes estándar ROS del *package std_msgs* para intercambiar información (Float32, Int8, Int16, Bool), así como de mensajes específicos de la geometría del *package geometry_msgs* (Point32, Pose2D).

Todas las funciones de *callback* se basan en la activación de *topics* para la lectura de los comandos, evitando desbordamientos de mensajes con información innecesaria. Por ejemplo, la función para detener los motores está siempre disponible como una interrupción externa. Como lenguaje de programación para el microcontrolador ATmega328p se utiliza C/C++, combinado con el cliente ROS y las bibliotecas OMNI-3MD. El algoritmo 1 ilustra el código *firmware* residente del *TraxBot*, que se ejecuta en la placa Arduino Uno, como se muestra en la Fig. 7. Téngase en cuenta que ROS proporciona diversos tipos de mensajes específicos para sensores que pueden ser utilizados directamente en los *topics* de cada componente del controlador del *TraxBot*.

El servidor de ROS se ejecuta en el *netbook*, en el cual, el nodo de conexión del *stack rosserial* permite iniciar la comunicación con el *TraxBot*, proporcionando así los *topics* disponibles para suscribir y publicar. Una de las muchas herramientas de ROS, *rxgraph*, se utiliza para permitir la monitorización en tiempo real de los nodos disponibles, así como los *topics* y mensajes de cada nodo.

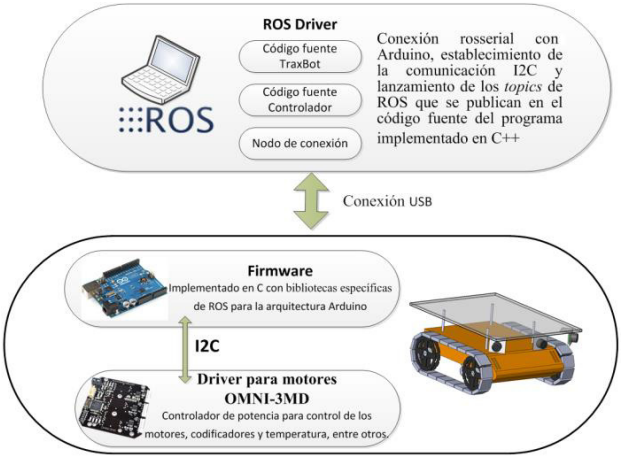

Figura 7: Estructura del controlador ROS del robot.

Algoritmo 1. *Firmware* residente en el Robot/Arduino.

```
#Omni3MD  //principales funciones de control de los motores
#EEPROM  //almacenamiento de especificaciones: robot ID,…
#Robot  //adquisición de sensores sónar, calibración, ganancias PID
#RosserialCommlibrary  //protocolo de comunicación serie
#Bibliotecas estándar
SetupFunctions(); // ganancias PID de los motores, puertos, escala de los
encoders, conexión I2C, ...
Functions():
    EncodersReads()
        Leemos los pulsos de los encoder 1 y 2;
    SonarsReads()
        Leemos los rangos de los sónares 1, 2 y 3;
    sendRobotInfo()
        Leemos el robot ID de la EEPROM;
        Leemos la temperatura interna de la placa;
        Leemos la versión de firmware del driver Omni-3MD;
        Leemos la tensión de las baterías del TraxBot;
        Leemos la versión de firmware del Arduino;
Main loop():
    Switch (action):
        Auto-calibración de los motores a través del Omni-3MD;
        Establecemos las ganancias del PID;
        Establecemos el prescaler de los encoders;
        Establecemos los valores iniciales de los encoders;
        Obtenemos la información del Robot;
        Realizamos una única lectura de los encoders;
        Realizamos una única lectura de los sónares;
        Movimiento lineal de los motores a través del controlador PID;
        Movimiento lineal de los motores;
        Parar los motores;
        Resetear los motores;
```

En la Fig. 8, se muestra un diagrama de interconexión de los nodos de ROS. Se puede observar que el controlador del *TraxBot* está conectado al nodo *TraxBot_DriverTest*, un programa desarrollado en ROS, que pone a prueba todos los *topics* y funciones de la plataforma. El nodo *TraxBot_DriverTest* verifica todas las funcionalidades implementadas en el *TraxBot*: testea el *firmware* y el movimiento del robot mediante el envío de comandos de velocidad, y lee de forma continua las actualizaciones de odometría proporcionadas por los *encoders* ópticos.

5.1. Controlador: características y potencialidades

En la última sección hemos presentado un controlador de código abierto que permite la integración en ROS de una plataforma robótica educativa basada en Arduino, que puede ser utilizada como punto de partida para generalizarse fácilmente a una amplia variedad de robots, también sobre la base de placas *Arduino.*

Este *driver* de ROS utiliza *rosserial* para interconectar la placa Arduino y ROS a través de una comunicación en serie. La característica más importante en *rosserial* consiste en hacer accesibles las bibliotecas de ROS en el código fuente de Arduino, con el fin de emular el lenguaje ROS directamente desde el código Arduino. Esto permite al *driver* publicar *topics* ROS con información diversa relacionada con el *TraxBot*, así como suscribirse a datos que provienen de los comportamientos de alto nivel y que se deben enviar al robot.

El código *firmware* para Arduino fue desarrollado teniendo en cuenta todos los componentes del robot y sus características que son necesarios para la operación. El robot proporciona información para leer los codificadores, temperatura, el establecimiento de los parámetros de las configuraciones iniciales del controlador PID y estado de la batería, entre otros. Todo esto es proporcionado directamente por el *driver* del motor OMNI-3MD, que transmite esta información a la placa Arduino mediante I2C.

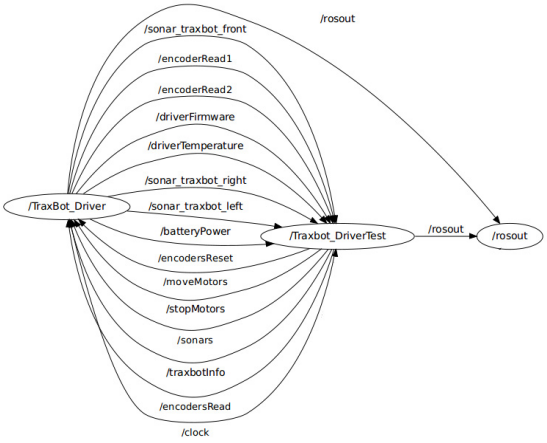

Figura 8. *Topics rxgraph* y nodos proporcionados por el controlador ROS del *TraxBot*. Un ejemplo ilustrativo de aplicación.

En el lado de ROS/*netbook*, cuando se recibe nueva información del robot (es decir, desde el Arduino Uno), esta es publicado por el *driver* de ROS desarrollado, en un *topic* apropiado, y está disponible para cualquier otra aplicación que ejecute ROS, es decir, un nodo de ROS. Los nodos pueden suscribirse a la información a través del *topic* apropiado, por ejemplo, pueden suscribirse al *topic* "/BatteryPower" para decidir cuándo el robot debe ser conducido a una estación de carga. Del mismo modo, cada vez que se programa una nueva aplicación, podemos proporcionar datos al robot, por ejemplo, comandos de velocidad para conducir el *TraxBot*, mediante la publicación en el *topic* correcto "/moveMotors".

Además de controlar el robot, el *driver* de ROS permite la interfaz con herramientas de ROS para el procesamiento y análisis de datos de la plataforma *TraxBot*, como la visualización en 3D (*rviz*), el registro en tiempo real de los experimentos llevados a cabo con los robots y la posibilidad de reproducirlos posteriormente (*rosbag/rxbag*), la representación gráfica de los datos (*rxplot*) y la visualización el toda la estructura de red de ROS (*rxgraph*). Adicionalmente, el *driver* permite la interfaz con otros sensores diferentes; interactuar con algoritmos programados en ROS, ya sea por el usuario o los ya existentes; experimentos con equipos de robots y la interfaz con el mundo virtual y compañeros de equipo virtuales. Todo esto se muestra más adelante.

Como resultado, el tiempo total dedicado al desarrollo de una solución robótica se reduce significativamente y por lo tanto el controlador proporcionado representa una valiosa herramienta que permite la rápida creación de prototipos y abre una puerta de entrada en el mundo de ROS. La interacción entre los programas de alto nivel y los recursos disponibles en la plataforma TraxBot permite la abstracción del *hardware* y la posibilidad de utilizar las interfaces estándar de cualquier plataforma robótica móvil integrada con la arquitectura ROS. Un ejemplo importante de aplicación es que con el controlador que aquí se presenta, el mismo código se puede utilizar tanto para manejar robots reales como para controlar agentes virtuales simulados ejecutándose en Stage o Gazebo. Además, la comunicación entre los agentes reales y virtuales en equipos de robots es completamente transparente, ya que todos se ejecutan en el *middleware* ROS.

Además, siempre es posible monitorizar, registrar y depurar en cualquier momento toda la información intercambiada y los nodos específicos que se ejecutan en la red.

6. Evaluación Experimental

Con el fin de evaluar experimentalmente la plataforma aquí propuesta, así como el controlador para ROS, se han llevado a cabo diferentes experimentos en un escenario de laboratorio, utilizando plataformas *TraxBot* tanto reales como virtuales. Los robots simulados fueron creados en *Stage*, que proporciona opciones esenciales como la información sobre la posición real y la odometría de los robots virtuales. Ambos, los robots reales y los virtuales se controlan mediante comandos de velocidad a través de un *topic* de ROS.

En un trabajo anterior (Araújo *et al.*, 2012), ya se realizó un análisis preliminar de la odometría y la precisión de la detección de los sónares del robot. Las siguientes pruebas consisten en el análisis de la exactitud de la información de detección proporcionada por dos sensores sónar laterales montados en la parte superior del robot, que se comparan con un sensor láser de rango. Posteriormente, se lleva a cabo un experimento de navegación, donde se replica el movimiento del robot en el simulador *Stage*. A continuación, se calcula la sobrecarga del controlador midiendo los tiempos de los mensajes pasados entre las tres capas de nuestro robot, y, por último, llevamos a cabo experimentos con un equipo de agentes reales y virtuales para comprobar la interacción de los diferentes agentes y el funcionamiento con evitación de obstáculos.

6.1. Detección y Mapeo

Para llevar a cabo este experimento, se equipó temporalmente a la plataforma *TraxBot* con un láser de rango (LRF) Hokuyo URG-04LX (Kneip *et al.*, 2009) para comparar su desempeño contra el conjunto de sónares ultrasónicos disponibles en la plataforma, en una tarea de mapeo y para mostrar la flexibilidad a la hora de integrar nuevos sensores en el robot gracias al *driver* de ROS. El láser Hokuyo URG-04LX produce un barrido horizontal en un rango de 240°, con una resolución angular de 0,36° y a una tasa de 10Hz. El rango teórico es de entre 20 y 4095 mm con un error de ± 2%.

Con el objetivo de probar el rendimiento de los sónares, se construyó un escenario de 2 m por 1,6 m en forma de L (Fig. 10). Para realizar esta prueba, se utilizaron sólo dos sónares laterales colocados a 45 grados, ya que el objetivo de esta prueba era evaluar la exactitud de los sónares, haciendo un mapeo del escenario basado en la combinación de la información de odometría y las lecturas de los sónares. El sónar frontal, se utiliza por lo general para la evitación reactiva de obstáculos, dado que en situaciones de mapeo, la proyección de paredes y obstáculos detectados por el cono del sónar, se ubicará siempre enfrente del robot, lo que es muy poco fiable en la mayoría de situaciones, tal como se muestra por ejemplo en la Fig. 9. En esta prueba, el movimiento del robot se basa únicamente en la *odometría*. En la Fig. 10a se puede ver en el primer movimiento rectilíneo, que las lecturas de los sónares son estables (puntos rojos) y coincidentes con los límites reales del escenario (línea azul). Los puntos verdes representan el punto medio del haz acústico sónar durante las rotaciones.

Cuando hay una rotación de 90 grados, surgen algunos problemas debido a que el cono del haz del sónar tiene una abertura de aproximadamente 36 grados, presentando así una pérdida de resolución (Fig. 9a). En el caso de la Fig. 10b, se utilizó el sensor láser Hokuyo para llevar a cabo la misma prueba. La apertura del láser se fijó en 180 grados con una tasa de 512 muestras por lectura. Es posible observar algunas discrepancias en algunas lecturas sobre todo al final del movimiento debidas al error de posición acumulado en la odometría durante el movimiento.

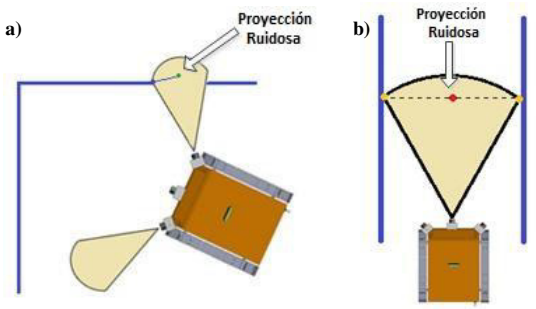

Figura 9: Situaciones donde hay lecturas ruidosas. a) Problema en las rotaciones utilizando sónares laterales; b) Esta figura ilustra por qué el sónar frontal no se utiliza para el mapeo.

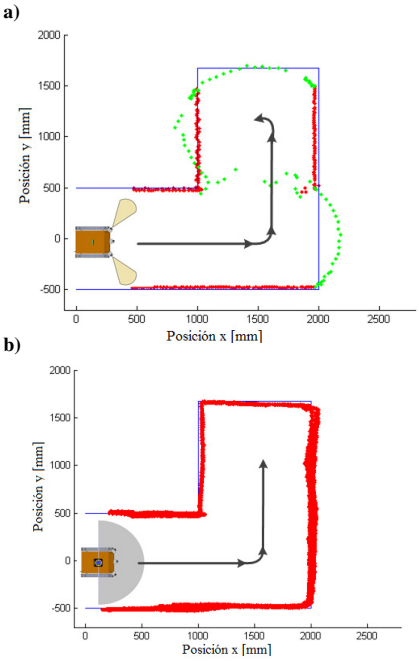

Figura 10: Test de mapeo. a) Sónares b) Láser.

6.2. Test del Controlador ROS

Para testear el *driver* para ROS se ha realizado una prueba consistente en realizar una trayectoria en escalera en un mapa cuadriculado de 400×400 mm, tal como se muestra en la Fig. 11a. En este "test en escalera", se utilizó un algoritmo de navegación punto a punto independiente, utilizando comandos de posición *(x, y, theta)*. En el desarrollo de este algoritmo, no se utiliza la navigation stack. En su lugar, se ha programado un nodo de ROS en C++, utilizando la biblioteca cliente roscpp, implementando un comportamiento de navegación simple. Este nodo, una vez en ejecución, y tras un proceso de compilación, se suscribe al topic con datos de sonar y publica comandos de velocidad en el topic apropiado para conducir el robot.

Se añadieron al mapa dos cajas de diferentes dimensiones como puntos de referencia. El objetivo principal del experimento era probar la conectividad entre los distintos nodos de la red. Se utilizó una red compuesta por dos ordenadores para distribuir la carga de procesamiento, liberando al *netbook* eeePC, y replicando el experimento simultáneamente en el simulador *Stage*. El nodo ROS master *(roscore)* y el nodo *stageros* se ejecutaban en un ordenador de sobremesa, y el eeePC se colocó en la parte superior del *TraxBot* que ejecutaba el *driver* de ROS (*TraxBot_Driver*) y el nodo con el algoritmo (*TraxBot_DriverTest*).

Mientras se hacía el "test en escalera", el *TraxBot* publicaba las actualizaciones de la odometría en un *topic* de ROS en tiempo real. A este *topic* se encontraba suscrito el nodo de *Stage*, encargado de procesar los mensajes y de replicar los movimientos en el mundo virtual usando un robot virtual (en la pantalla del ordenador), como se muestra en la Fig. 11. El retraso de todo este proceso es menor de un segundo.

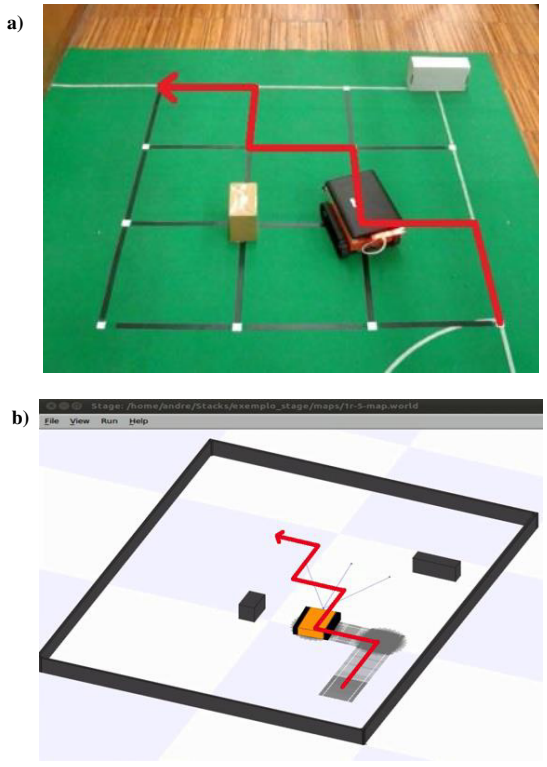

Figura 11: Test del *driver* de ROS.
a) Test "en escalera" real; b) Simulación Stage.

6.3. Tiempo de respuesta del Controlador ROS

Para el análisis de la sobrecarga que supone la utilización del *driver* de ROS, se midió el tiempo de respuesta de los mensajes intercambiados entre las tres capas del sistema (OMNI-3MD, Arduino Uno y ROS).

Se puede observar en la Fig. 12 que el tiempo de respuesta total entre el *driver* de los motores OMNI-3MD y ROS, aumenta con el número de mensajes, tal como era de esperar. Sin embargo, los valores medidos son insignificantes cuando se consideran aplicaciones con robots reales. 36ms, un retardo muy residual que no es crítico cuando no existen restricciones de tiempo real.

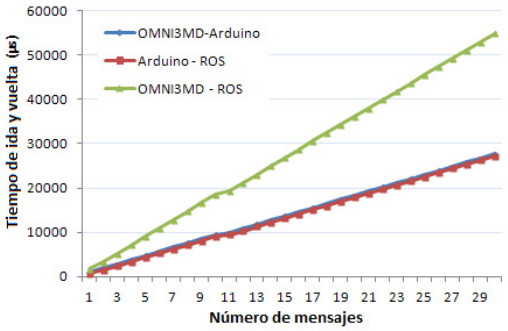

Figura 12: Tiempo de respuesta de los mensajes del controlador ROS.

Por ejemplo, el número medio de mensajes intercambiados por segundo en el ensayo experimental anterior es de 20, lo que corresponde a un tiempo de respuesta de Sin embargo, como aspecto negativo, también se llegó a la conclusión de que el microprocesador ATmega328P utilizado en el robot presenta una limitación en términos de memoria SRAM.

Cuando se ejecuta el *driver* de ROS, sólo quedan libres 283 bytes de memoria en el Arduino Uno, lo que significa que para los *topics* estándar (mensajes float32 + encabezados del *topic*), sólo se pueden añadir 7 *topics* más para extensiones y el buffer de mensajes se limita a 70 mensajes estándar.

6.4. Equipos de Robots Virtuales y Reales

La simulación de múltiples robots es posible en ROS a través de un contenedor para *Stage* que soporta un gran número de robots interactuando dependiendo de la complejidad computacional del algoritmo. Sin embargo, la posibilidad de ejecutar nodos en múltiples CPUs en la misma red aumenta considerablemente los límites de tamaño del equipo, eliminando la sobrecarga de procesamiento de una única máquina.

En este estudio experimental, se desplegaron dos *TraxBots* en un espacio cerrado, ejecutando un algoritmo reactivo de deambulación, tal como se ilustra en la Fig. 13. Esta prueba se realizó para verificar la coordinación entre múltiples robots. En este caso, se utilizó un equipo mixto de robots reales y virtuales, para demostrar las ideas presentadas en la sección 5.

Los dos *TraxBots* reales fueron replicados en *Stage* (Fig. 13b), y se añadió un *TraxBot* virtual adicional, con exactamente el mismo algoritmo reactivo. Esto puede ser visto como una característica interesante para los programadores, en la que con ajustes menores, es posible utilizar el mismo código, ya sea para robots reales o virtuales. Las líneas azules proyectadas delante de los robots corresponden a lecturas a escala real de los tres sónares ultrasónicos instalados en la plataforma *TraxBot*.

Los tres robots fueron capaces de coordinarse en el medio sin chocar entre sí a través de un comportamiento reactivo, como se muestra en el vídeo (Vídeo, 2013) de los experimentos.

7. Conclusiones y trabajo futuro

En este trabajo se presenta el desarrollo y la evaluación experimental de una plataforma robótica compacta y una solución para la integración de la plataforma en ROS, mediante el desarrollo de un controlador para *firmware* del robot.

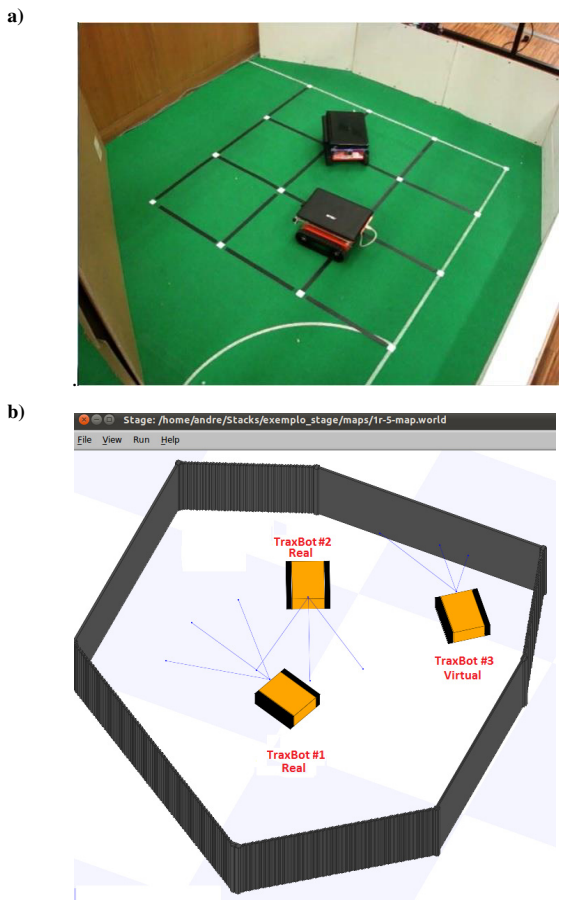

Figura 13: Test del controlador. a) Dos robots con navegación reactiva; b) Mezcla de robots reales y virtuales.

Se han mostrado las ventajas de la integración de la plataforma con el *middleware* ROS, permitiendo el uso de una amplia gama de herramientas y reduciendo el tiempo de desarrollo a través de la reutilización de código.

Más concretamente, en este trabajo, el tiempo de desarrollo se ha reducido de varias maneras. En primer lugar, en el desarrollo del *driver* de ROS mediante el uso del *stack rosserial* existente, que es el responsable de la comunicación serie entre ROS y la placa Arduino. En segundo lugar, mediante el uso de los ya definidos mensajes estándar de ROS, para publicar y suscribirse a datos desde/hacia el robot. En tercer lugar, mediante la integración del LRF en el robot, usando el *stack laser_drivers*, y más específicamente, el *package hokuyo_node* (sección 6.1). En cuarto lugar, mediante la interacción con el mundo virtual, usando un simulador multi-robot ya integrado en ROS (*stack stage*). Por último, mediante el uso en todos los experimentos de todo tipo de herramientas de línea de comandos que ROS ofrece en su entorno, como *rosbag*, *rviz*, *rxgraph*, *rostopic*, etc.

Además de proporcionar acceso a todas las herramientas de ROS, el *driver* simplifica el desarrollo de programas ya que proporciona abstracción de *hardware*, permitiendo controlar fácilmente la plataforma. También permite la ampliación y la integración de todo tipo de sensores, y permite la cooperación y la coordinación multi-robot a través de la operación en una red ROS, permitiendo equipos reales de robots homogéneos y heterogéneos, así como equipos híbridos de agentes reales y virtuales, ejecutando el mismo código. Los resultados obtenidos en los experimentos demuestran todas estas características y la insignificante sobrecarga que supone el uso del *driver* de ROS. Este *driver* está disponible para su descarga (TraxBot Driver, 2014) y pretende ser un aporte para las comunidades de ROS y Arduino, y ayudar a impulsar el desarrollo de robots personalizados.

Siendo parte de nuestro trabajo en curso, se está desarrollando un protocolo de comunicación serie personalizado que puede reemplazar al nodo *rosserial*, superando así la limitación que supone el retardo en el intercambio de los mensajes. Los detalles de este desarrollo están reportados en nuestro trabajo específico (Araújo *et al.*, 2013), que se centra fundamentalmente en la extensión del *driver* de ROS para diferentes plataformas basadas en Arduino, más allá del *TraxBot*.

Como trabajo futuro, tenemos varios planes en marcha: integrar la comunicación ZigBee directamente a través de ROS, dar cabida a diferentes tipos de robots en el equipo, para poder desarrollar experimentos heterogéneos; integrar el *stack* de navegación de ROS con la adición de un láser Hokuyo, integrar un sensor Kinect para dotar al robot con la capacidad de realizar 3D SLAM, desarrollar un nodo de tele-operación, y utilizar un teléfono Android para aprovechar las ventajas de su capacidad de comunicación, así como de sus sensores internos (brújula electrónica, GPS, acelerómetros, cámara, sensor de luz, etc).

English Summary

Development of a compact mobile robot integrated in ROS middleware.

Abstract

This paper presents the *TraxBot* robot and its full integration in the Robotic Operating System (ROS). The *TraxBot* is a compact mobile robotic platform developed and assembled at the Institute of Systems and Robots (ISR) Coimbra. The goal in this work is to drastically decrease the development time, providing hardware abstraction and intuitive operation modes, allowing researchers to focus in their main research motivations, *e.g.*, search and rescue, multi-robot surveillance or swarm robotics. The potentialities of the *TraxBot* are described which, combined with the ROS driver developed, provide several tools for data analysis and easiness of interaction between multiple robots, sensors and tele-operation devices. To validate the approach, diverse experimental tests using real and virtual simulated robots were conducted.

Keywords:
ROS, mobile robot, Arduino, embedded system, design, assembling and testing.

Agradecimientos

Este trabajo fue financiado por las becas de doctorado (SFRH/BD/64426/2009) y (SFRH/BD/73382/2010) otorgadas por la Fundación Portuguesa para la Ciencia y la Tecnología (FCT), el Instituto de Sistemas y Robótica (ISR) y RoboCorp en el Instituto de Ingeniería de Coimbra (ISEC) y por el proyecto de investigación CHOPIN (PTDC/EEA-CRO/119000/2010) también en la financiación ordinaria de FCT.

Referencias

Ahn S, Lee J, Lim K, Ko H, Kwon Y, Kim H (2006) Requirements to UPnP for Robot Middleware. *Proc. Int. Conf. on Intelligent Robots and Systems (IROS 2006)*, Beijing, China, 4716-4721.

Araújo A, Portugal D, Couceiro M, Figueiredo C, Rocha RP (2012) TraxBot: Assembling and Programming of a Mobile Robotic Platform. Proc. *Int. Conf. on Agents and Artificial Intelligence (ICAART 2012)*, Vilamoura, Algarve, Portugal, 301-304.

Araújo A, Portugal D, Couceiro M, Rocha RP (2013) Integrating Arduino-based Educational Mobile Robots in ROS. Proc. *Int. Conf. on Autonomous Robot Systems and Competitions (Robotica 2013)*, Lisboa, Portugal, 8-13.

Arduino Uno (2010) http://arduino.cc/en/Main/ArduinoBoardUno. Último acceso: 13.02.2014.

Bagnall B (2007) *Maximum LEGO NXT: Building Robots with Java Brains*. Variant Press.

BRICS (2014) http://best-of-robotics.org Último acceso: 13.02.2014.

Brugali D, Scandurra P (2009) Component-based Robotic Engineering (Part I): Reusable building blocks. *IEEE Robotics and Automation Magazine*, 16 (4), 84-96.

Colot A (2010) *K-Team Hemisson Manual*, K-Team S.A., Yverdon-les-Bains, Switzerland, http://ftp.k-team.com/hemisson/Manuel_En/Manual_Hemisson.pdf. Último acceso: 13.02.2014.

Couceiro MS, Rocha RP, Ferreira NM (2011) A Novel Multi-Robot Exploration Approach based on Particle Swarm Optimization Algorithms. Proc. *Int. Symp. on Safety, Security, and Rescue Robotics (SSRR 2011)*, Kyoto, Japan, 327-332.

Cummins J, Azhar MQ, Sklar E (2008) Using Surveyor SRV-1 Robots to Motivate CS1 Students. *Proc. AAAI 2008 Artificial Intelligence Education Colloquium*, Chicago, Illinois, USA, 23-27.

Data Translation (2006) *Using Quadrature Encoders/Decoders for X/Y Positioning and Rotation*, January, 2006. http://goo.gl/GGI9lK. Último acceso: 13.02.2014.

Gazebo (2014) http://gazebosim.org Último acceso: 13.02.2014.

Gerkey B, Vaughan R, Howard A (2003) The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. Proc. *Int. Conf. on Advanced Robotics (ICAR 2003)*, Coimbra, Portugal, 317-323.

Grisetti G, Stachniss C, Burgard W (2007) Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters. *IEEE Transactions on Robotics*, 23 (1), 34-46.

IdMind, Engenharia de Sistemas, Lda. (2005) *Manual de construção Robô Circular GT*, http://aprobotica.com.sapo.pt/ManualCircularGT.pdf

Jones JL (2006). Robots at the tipping point: the road to iRobot Roomba. *IEEE Robotics & Automation Magazine*, 13 (1), 76-78.

Kneip L, Tâche F, Caprari GR (2009) Characterization of the compact Hokuyo URG-04LX 2D laser range scanner. Proc. *IEEE International Conference on Robotics and Automation (ICRA 2009)*, Kobe, Japan, 1447-1454.

Kuipers M (2009) Localization with the iRobot Create. Proc. *47th Annual Southeast Regional Conference ACM (ACM-SE),* Clemson, South Carolina, USA, 1-3.

Machado Santos J, Portugal D, Rocha RP (2013) An Evaluation of 2D SLAM Techniques Available in Robot Operating System. Proc. *Int. Symp. on Safety, Security, and Rescue Robotics (SSRR 2013)*, Linköping, Sweden.

Marder-Eppstein E, Berger E, Fully T, Gerkey B, Konolige K (2010) The Office Marathon: Robust Navigation in an Indoor Office Environment. Proc. *Int. Conf. on Robotics and Automation (ICRA 2010)*, Anchorage, Alaska, 300-307.

MARIE (2014) http://marie.sourceforge.net Último acceso: 13.02.2014.

MB1300 XL-MaxSonar®-AE0™ Datasheet (2005) http://www.maxbotix.com/documents/MB1200-MB1300_Datasheet.pdf. Último acceso: 13.02.2014.

Mohamed N, Al-Jaroodi J, Jawhar I (2008) Middleware for Robotics: A Survey. Proc. *Int. Conf. on Robotics, Automation, and Mechatronics (RAM 2008)*, Chengdu, China, 736-742.

MRPT (2014) http://www.mrpt.org Último acceso: 13.02.2014.

OMNI-3MD *Manual Bot'n Roll* (2011) http://botnroll.com/omni3md/downloads/OMNI-3MD_Software_Manual(17-10-2013).pdf Último acceso: 13.02.2014.

OpenCV (2014) http://opencv.org Último acceso: 13.02.2014.

ORCA (2014) http://orca-robotics.sourceforge.net Último acceso: 13.02.2014.

OROCOS (2014) http://orocos.org Último acceso: 13.02.2014.

Petrina AM (2011) Advances in Robotics. In *Automatic Documentation and Mathematical Linguistics*, Allerton Press, 45 (2), 43–57.

Player/Stage (2014) http://playerstage.sourceforge.net Último acceso: 13.02.2014.

Portugal D, Rocha RP (2011) On the Performance and Scalability of Multi-Robot Patrolling Algorithms. Proc. *Int. Symp. on Safety, Security, and Rescue Robotics (SSRR 2011)*, Kyoto, Japan, 50-55.

Portugal D, Rocha RP (2012) Decision Methods for Distributed Multi-Robot Patrol. Proc. *Int. Symp. on Safety, Security, and Rescue Robotics (SSRR 2012)*, College Station, Texas, USA.

Quigley M, Gerkey B, Conley K, Faust J, Foote T, Leibs J, Berger E, Wheeler R, Ng AY (2009) ROS: an open-source Robot Operating System. Open-Source Software Workshop, Proc. *Int. Conf. on Robotics and Automation (ICRA 2009)*, Kobe, Japan.

Roomba (2014) http://wiki.ros.org/Robots/Roomba Último acceso: 13.02.2014.

Root M (2010) *The Tab Battery Book: An In-Depth Guide to Construction, Design, and Use*. McGraw-Hill/Tab Electronics, ISBN: 978-0071739900.

Rosserial (2014) http://wiki.ros.org/rosserial Último acceso: 13.02.2014.

Rusu R, Cousins S (2011) 3D is here: Point Cloud Library (PCL). Proc. *Int. Conf. on Robotics and Automation (ICRA 2011),* Shanghai, China, 1-4.

SAR - Soluções de Automação e Robótica (2010) Manual Bot'n Roll ONE C. Guimarães, Portugal, http://botnroll.com/onec/downloads/Manual%20Bot'n%20Roll%20ONE%20C%20EN.pdf. Último acceso: 13.02.2014.

Schlegel C, Steck C, Lotz A (2011) Model-driven software development in robotics: Communication patterns as key for a robotics component model. *Introduction to Modern Robotics,* iConcept Press Ltd.

SMARTsoft (2014) http://smart-robotics.sourceforge.net Último acceso: 13.02.2014.

TraxBot Driver (2014) http://wiki.ros.org/traxbot_robot Último acceso: 13.02.2014.

Video (2013) http://www.isr.uc.pt/~aaraujo/videos/Iberoamericana2013

Wunsche B, Chen I, MacDonald B (2009) Mixed Reality Simulation for Mobile Robots. Proc. *Int. Conf. on Robotics and Automation (ICRA 2009)*, Kobe, Japan, 232-237.

XBee® Multipoint RF Modules Datasheet (2006), http://www.digi.com/pdf/ds_xbeemultipointmodules.pdf. Último acceso: 13.02.2014.

YARP (2014) http://wiki.icub.org/yarp Último acceso: 13.02.2014.