Tecnura
ISSN: 0123-921X
tecnura@udistrital.edu.co
Universidad Distrital Francisco José de Caldas
Colombia

Martínez Sarmiento, Fredy Hernán; Giral Ramírez, Diego Armando
OpenRRArch: una arquitectura abierta, robusta y confiable para el control de robots autónomos
Tecnura, vol. 21, núm. 51, enero-marzo, 2017, pp. 96-104
Universidad Distrital Francisco José de Caldas
Bogotá, Colombia

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Tecnura

http://revistas.udistrital.edu.co/ojs/index.php/Tecnura/issue/view/818

DOI: http://dx.doi.org/10.14483/udistrital.jour.tecnura.2017.1.a07

INVESTIGACIÓN

OpenRRArch: una arquitectura abierta, robusta y confiable para el control de robots autónomos

OpenRRArch: An open, robust and reliable architecture for controlling autonomous robots

Fredy Hernán Martínez Sarmiento[1], Diego Armando Giral Ramírez[2]

Fecha de recepción: 16 de febrero de 2016 **Fecha de aceptación:** 23 de noviembre de 2016

Cómo citar: Martínez S., F.H. y Giral R. D.A. (2017). OpenRRArch: una arquitectura abierta, robusta y confiable para el control de robots autónomos. *Revista Tecnura, 21*(51), 96-104.

Resumen

Contexto: Los sistemas de control y navegación de robots autónomos constituyen un dinámico campo de investigación en robótica. Los esquemas y estrategias propuestos como posible solución a problemas se evalúan siempre sobre prototipos de laboratorio a fin de determinar su desempeño real. Se propone en este artículo una arquitectura para el diseño y desarrollo de sistemas robóticos, particularmente sistemas autónomos multirrobot, que facilite el trabajo en laboratorio gracias a una arquitectura abierta con características de robustez y confiabilidad.

Método: La arquitectura se soporta en herramientas *hardware* y *software open source*. La estrategia de operación y comunicación se caracteriza por un bajo consumo de recursos, tanto en procesamiento como en comunicación, y operación en tiempo real. Se utiliza Linux como plataforma de desarrollo y sistema operativo, en particular para el esquema de comunicación, y los sistemas embebidos sobre un procesador de 32 bits con set de instrucciones de 16 bit (no es ARM, pero con arquitectura Harvard) corriendo a 80 MHz (el LX106 de Tensilica Xtensa) para la implementación de los agentes. Dadas las herramientas utilizadas, la solución también demuestra ser eficiente y económica.

Resultados: La arquitectura ha sido aplicada exitosamente en la implementación de una estrategia de navegación para un conjunto de pequeños robots autónomos. A un conjunto de robots se les proporciona capacidad de comunicación WiFi, capacidad mínima de localización del ambiente (detección de obstáculos) y un algoritmo de navegación basado en tamaños poblacionales (imitando el Quorum Sensing Bacterial). El sistema se implementa con gran facilidad, demostrando no sólo la viabilidad de la estrategia de navegación, sino también la versatilidad, robustez y escalabilidad de la arquitectura OpenRRArch.

Conclusiones: La arquitectura propuesta constituye una solución para la construcción de sistemas de control distribuidos, en particular sistemas robóticos multiagente. Ésta permite una rápida implementación (a bajo costo y con alto desempeño) de sistemas con capacidad de cooperación y comunicación en tiempo real. La arquitectura permite la integración de agentes con diferentes capacidades de procesamiento, lo cual permite adaptación a las necesidades de la tarea. Las funciones de cada uno de los agentes

1 Ingeniero electricista, especialista en Gestión de Proyectos de Ingeniería, candidato a doctor en Ingeniería – Sistemas y Computación. Docente de la Universidad Distrital Francisco José de Caldas. Bogotá, Colombia. Contacto: fhmartinezs@udistrital.edu.co

2 Ingeniero eléctrico, magíster en Ingeniería Eléctrica. Docente de la Universidad Distrital Francisco José de Caldas. Bogotá, Colombia. Contacto: dagiralr@correo.udistrital.edu.co

pueden reducirse o aumentarse a partir de su código, o seleccionando otro *hardware* embebido.

Palabras clave: control, Linux, Open-source, robots, sistemas embebidos.

Abstract

Context: The control and navigation systems of autonomous robots constitute a dynamic field of research in robotics. The possible solutions to problems are often evaluated with laboratory prototypes in order to determine their real performance. We propose in this paper an architecture for the design and development of robotic systems (particularly autonomous multi-robot Systems) that facilitate the work in laboratory due to an open robust and reliable architecture.

Method: The architecture is supported in open source hardware and software. The operation and communication strategy is characterized by a low consumption of resources, both in processing and communication, and real-time operation. As develpement platform and Operating System, we used Linux: the communication scheme and the embedded systems run on a 32-bit processor, with a 16-bit instruction set (not ARM, but with Harvard architecture), at 80 MHz (Tensilica Xtensa LX106) for the implementation of the agents. The tools used allow the solution to be bothe efficient and inexpensive.

Results: The architecture has been successfully applied in the implementation of a strategy of navigation for a set of small autonomous robots. A set of robots were provided with wireless communication capability, minimum capacity of environmental sensing (obstacle detection) and a navigation algorithm based on population sizes (mimicking the bacterial Quorum Sensing). The system is implemented with great ease, demonstrating both the viability of the navigation strategy and the versatility, robustness and scalability of the OpenRRArch architecture.

Conclusions: The proposed architecture constitutes a solution for the construction of distributed control systems, in particular multi-agent robotic systems. It enables the rapid, low-cost and high-performance implementation of systems with real-time cooperation and communication capabilities. The architecture allows the integration of agents with different processing capabilities, which allows adaptation to the needs of the task. The functions of each agent can be reduced or increased from their code, and/or by selecting other embedded hardware.

Keywords: *Control, Embedded Systems, Linux, Open-source, Robots*.

INTRODUCCIÓN

El problema de encontrar una ruta de navegación para un robot móvil autónomo (o varios robots móviles autónomos en el caso de enjambres de robots) consiste en hacer que el agente (un objeto real con dimensiones físicas) encuentre y siga un camino que le permita moverse desde un punto de origen hasta un punto de destino (configuración deseada) respetando las restricciones impuestas por la tarea o por el ambiente (obstáculos, espacio libre para movimientos, puntos intermedios a visitar, y costos máximos a incurrir en la tarea) (Teatro, Eklund, y Milman, 2014). Este sigue siendo un problema abierto de investigación en robótica (Jacinto, Giral y Martínez, 2016; Oral y Polat, 2016).

En este artículo se presenta la propuesta de diseño de sistemas colectivos basados en OpenRRArch (arquitectura abierta, robusta y confiable para el control de agentes autónomos), esquema que encarna los principios de los sistemas multiagentes distribuidos (robustez mediante alta redundancia), abierta (*hardware* y *software open-source*), minimalista y de fácil integración (agentes simples e idénticos). El propósito de dicha arquitectura es el de permitir, entre otras funciones, el control de movimiento a bordo de pequeños robots autónomos, y la planificación de trayectorias a partir de lecturas locales con capacidad de comunicación *peer-to-peer*

(P2P). Esta arquitectura, sin embargo, puede ser utilizada en otro tipo de sistemas con configuraciones similares, como es el caso de las redes inalámbricas de sensores (*Wireless Sensor Networks*) (Kiwoong *et al.*, 2012; Lee, Cho y Lee, 2013).

Para la correcta navegación, en un esquema autónomo reactivo como el que se busca en esta investigación, cada agente (robot) debe localizar la información pertinente del medio (lo cual incluye, además de obstáculos y otras restricciones del ambiente, procesos de comunicación con agentes cercanos), realizar cálculos de su posición estimada respecto al destino, trazar una respuesta de movimiento, ejecutarla y verificar los resultados. Esta estrategia de navegación es fuertemente dependiente de la cantidad y calidad de información procesada y comunicada, razón por la cual se complementa con esquemas de filtrado de información inspirados en los procesos de atención de los seres vivos (Luna, Silva y Costa, 2016). El sistema (conformado por el agente, o los agentes en el caso general de sistemas colectivos) debe tener una estrategia de atención que le permita ejecutar diferentes algoritmos de acuerdo con el objetivo de la tarea. Este principio de diseño se utiliza mucho en los esquemas híbridos, y permite el desarrollo de sistemas adaptables y con alta velocidad de respuesta.

Otras características importantes incorporadas a la arquitectura propuesta incluyen alto desempeño tanto en *hardware* y *software*. En *hardware*, se pone especial cuidado en cuanto al consumo de energía (Elwahab, Nouali, Moussaoui y Derder, 2015), al esquema de comunicación IP-WSN (protocolo para la interconexión entre redes IP (*Internet Protocol*) y redes inalámbricas de sensores (*Wireless Sensor Networks*)) (Kiwoong *et al.*, 2012), y los recursos disponibles en los agentes (Kouicem *et al.*, 2014). En cuanto a *software*, se busca facilitar la construcción de colecciones de datos (Elwahab *et al.*, 2015), el uso de agentes específicos para la recolección de información (Carlson y Schrader, 2012), la fácil interacción entre agentes (Wenfeng, Junrong y Weiming, 2011), y simplificar al máximo la arquitectura misma de comunicación (Sirin, Parsia, y Hendler, 2005; Tari *et al.*, 2010; Yachir, Amirat, Chibani y Badache, 2012).

El grupo de investigación ARMOS ha trabajado por varios años en estos problemas. En cuanto a estrategias de planeación para robots móviles, ha evaluado el desempeño de diferentes estrategias geométricas (Contreras, Martínez y Martínez, 2015; Martínez, Jacinto y Martínez, 2014), ha planteado algoritmos de movimiento para sistemas reactivos (Espinosa, Castañeda y Martínez, 2015; Martínez y Delgado, 2010), ha desarrollado un amplio número de prototipos embebidos con diferentes capacidades de localización, comunicación y capacidad de procesamiento (Espinosa, Castañeda y Martínez, 2015; Montiel, Valderrama y Martínez, 2016), y ha formulado arquitecturas para la coordinación colectiva de robots autónomos. Este trabajo presenta la evolución actual de todos estos sistemas, estructurados alrededor de una única arquitectura que permite el desarrollo y la evaluación sistematizada de estrategias de control de movimiento para robots. La arquitectura como tal presenta una estructura modular diseñada por capas, lo que permite aislar problemas específicos durante la etapa de diseño.

El artículo se encuentra organizado de la siguiente forma. Primero, se formula el problema estudiado y se presentan algunos conceptos preliminares, el perfil funcional y algunas otras consideraciones de diseño. En "Metodología" se detalla completamente el diseño del sistema, incluyendo los criterios de selección y las especificaciones finales adoptadas. En "Resultados" se presenta la evaluación del desempeño observado en laboratorio del prototipo. Por último, se plantean las conclusiones.

FORMULACIÓN DEL PROBLEMA

La evaluación de estrategias de control distribuidas requiere de una plataforma soportada en agentes embebidos, con estructura modular y por capas, que permita una implementación transparente. La estructura por capas debe permitir separar e implementar diferentes funciones a diferentes niveles

de acuerdo con el nivel de desempeño requerido. Además, el sistema de comunicación debe ser eficiente, respaldado por un protocolo fácil de implementar, pero confiable. Los elementos a integrar en dicha arquitectura son:

1. Un enlace de comunicación P2P entre todos los agentes bajo algún protocolo estándar. Dicho protocolo debe ser fácil de implementar a nivel de *hardware* sobre cualquier dispositivo embebido (microcontroladores, CPLDs, FPGAs, DSPs, procesadores ARM y x86) mediante enlaces inalámbricos. Estas características son las esperadas en aplicaciones en donde se transmiten mensajes cortos durante cortos periodos de tiempo.
2. Fácil conexión a este esquema de comunicación por parte de todos los módulos embebidos del sistema multiagente. Cada módulo del sistema es, además de un elemento de comunicación, un nodo de control con capacidad de localización, procesamiento y actuación. Estos conforman la capa más baja a nivel de control, y de hecho ejecutan algoritmos directos de control. Dado que la capacidad de procesamiento es limitada, se espera que el esquema de comunicación no consuma sus recursos.
3. Una capa de control superior soportada en la convergencia del algoritmo de control, y que se ve reflejada en el comportamiento del sistema (de todos los módulos como un todo). Si bien en términos generales no se espera que exista una unidad de control central dada la naturaleza del sistema multiagente, la arquitectura sí debe permitir la inclusión de módulos embebidos de mayor capacidad de procesamiento para coordinar la comunicación. Estos módulos también deben desempeñar el papel de constructores de colecciones de datos (y, por tanto, integrarse fácilmente en un entorno IoT, *Internet of Things*), y localización/procesamiento de información del alto nivel (sensores inteligentes a partir de procesamiento de imágenes o sonido, por ejemplo). También pueden tener un algoritmo de operación más complejo que afecte el comportamiento global del sistema.

METODOLOGÍA

La arquitectura distribuida propuesta para el control de sistemas multiagentes autónomos (OpenRRArch) se muestra en la figura 1. El sistema permite una gran cantidad de módulos tanto en el nivel de bajo control como en el de alto control (por simplicidad solo se ilustra uno en cada caso). Sin embargo, lo normal es tener un solo módulo bróker en este último nivel (y, de hecho, debe existir por lo menos uno). Si se incluyen más módulos de nivel alto de control, estos se conectan al sistema como clientes MQTT (*MQ Telemetry Transport*). Los únicos elementos que son opcionales en la arquitectura son los terminales, los cuales se utilizan para visualizar por parte del usuario las variables del sistema.

En cuanto a funcionalidad, la arquitectura distingue dos capas: la de comunicación y la de aplicación. La primera se soporta en el estándar MQTT. Este es un estándar ISO (ISO/IEC PRF 20922) soportado en un protocolo de mensajería muy ligero que opera con métodos de suscripción-publicación. Está diseñado para aplicaciones de control en redes TCP/IP (*Transmission Control Protocol/Internet Protocol*) que requieren un código de implementación pequeño, como es el caso del *hardware* embebido. El protocolo requiere un bróker, el cual distribuye los mensajes en la red, y que en el caso de la figura 1 se implementa en un OS Linux Debian sobre una plataforma H3 Quad-core Cortex-A7 con GPU Mali400MP2. A este bróker suscriben y publican todos los demás módulos del sistema como clientes. El esquema de comunicación utilizado es *peer-to-peer*, con lo que cada módulo se puede comunicar directamente con cualquier otro módulo del sistema (siempre y cuando estén dentro del alcance del *router*).

La capa de aplicación incluye al conjunto de sensores y actuadores conectados a los diferentes módulos del sistema. En la prueba de navegación documentada en este artículo, el bróker cuenta con un sensor óptico (cámara digital) que se utiliza para identificar *landmarks* geométricos de colores en el

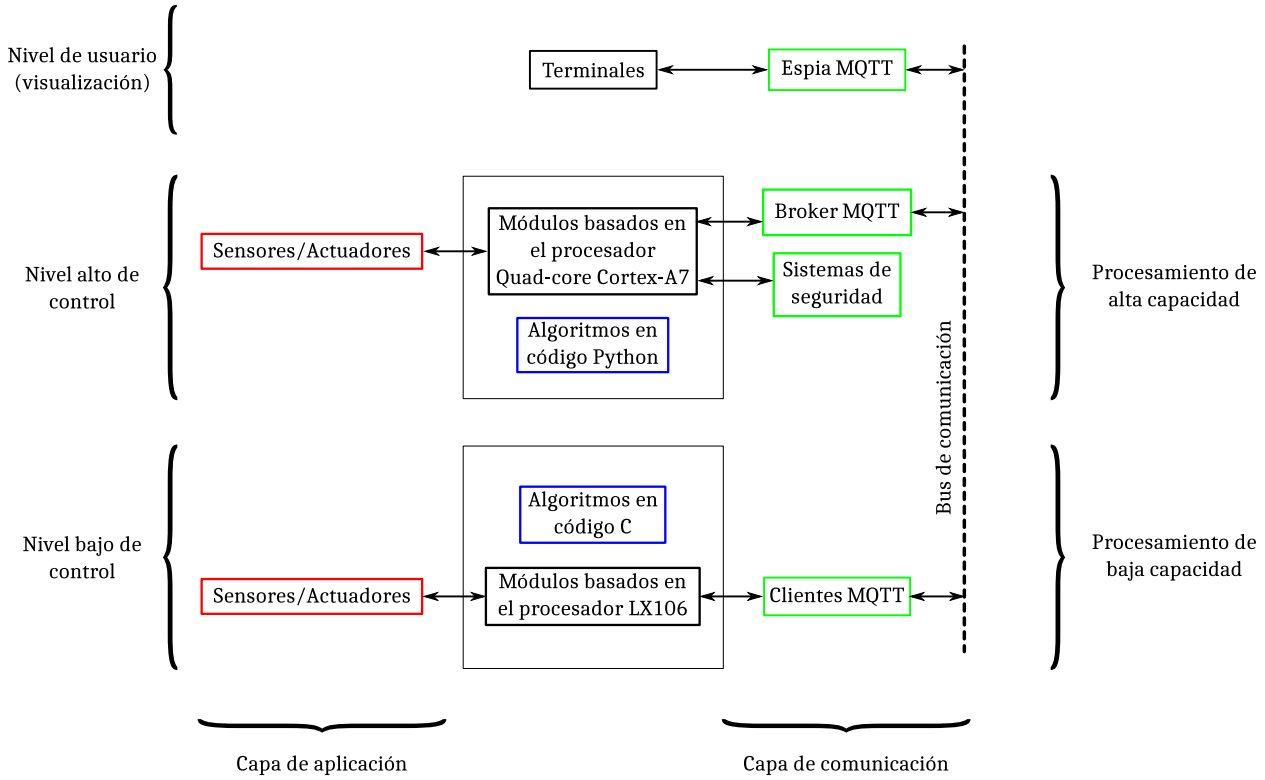

Figura 1. Diagrama general de la arquitectura del sistema

Fuente: elaboración propia.

ambiente. El código de procesamiento de las imágenes se escribió en Python con apoyo de OpenCV. También se implementaron otros dos módulos sobre plataformas microcontroladas (con el microcontrolador LX106), estos módulos se dotaron solamente con sensores de impacto. Los tres módulos poseen, cada uno, dos actuadores, dos servomotores que realizan el desplazamiento de los módulos en el ambiente. Todos los módulos poseen antenas y capacidad de conexión a la red WiFi.

La implementación de los algoritmos en ambos tipos de módulos es bastante sencilla. El microcontrolador LX106 ejecuta los códigos mucho más rápido que la plataforma Cortex-A7 (menos de 2 us frente a unos 70 ms). Python es interpretado, y la captura y procesamiento de imágenes es mucho más lenta que la lectura del pulsador digital. Sin embargo, esto no resulta de importancia en la aplicación debido al retardo en la acción de los actuadores (servomotores), que para desplazar al robot toman tiempos cercanos al segundo. No hay control de velocidad en los motores, el desplazamiento se logra haciéndolos rotar en un sentido u otro (A o B) o pararlos completamente, con lo cual se tienen siete estados de movimiento en el robot:

- Avanza (motor izquierdo en sentido A, y motor derecho en sentido B).
- Retrocede (motor izquierdo en sentido B, y motor derecho en sentido A).
- Gira a la derecha avanzando (motor izquierdo en sentido A, y motor derecho sin movimiento).
- Gira a la izquierda avanzando (motor izquierdo sin movimiento, y motor derecho en sentido A).
- Gira a la derecha retrocediendo (motor izquierdo sin movimiento, y motor derecho en sentido B).

- Gira a la izquierda retrocediendo (motor izquierdo en sentido B, y motor derecho sin movimiento).
- No se mueve (motor izquierdo sin movimiento, y motor derecho sin movimiento).

Esto es bastante útil para simular a nivel de estados discretos el comportamiento de los algoritmos de navegación (modelo híbrido del sistema).

RESULTADOS

En esta sección se presenta el análisis de desempeño de la arquitectura con una tarea de navegación real. La evaluación general pretende probar tres hipótesis: (1) que la efectividad de la estrategia de navegación puede analizarse independiente de la estrategia de implementación; (2) que la arquitectura OpenRRArch no altera el diseño del sistema; y finalmente (3) que el rendimiento del algoritmo de navegación se ve modificado por el *hardware* de la capa física, pero que estas variaciones no alteran su convergencia. Si estas tres hipótesis son correctas, entonces OpenRRArch es de hecho una excelente herramienta para estudiar el desempeño de estrategias de navegación.

La tarea de navegación utilizada para la evaluación de la arquitectura es bastante simple. Los módulos microcontrolados con solo sensores de impacto deben navegar el ambiente evitando obstáculos. Es decir, cuando uno de estos dos robots se estrella con un obstáculo (o un límite del ambiente), la orden de control indica que el robot debe girar a la derecha o izquierda (sentido elegido aleatoriamente) un ángulo aleatorio. La única excepción a este comportamiento es si otro robot reporta que encontró un obstáculo al mismo tiempo. En este caso los robots se deben poner de acuerdo para verificar si se han encontrado entre sí, caso en el que deben permanecer quietos. Si no se han encontrado, deben regresar a la navegación aleatoria.

El módulo con procesador Cortex-A7, bróker y con cámara digital (figura 2), debe buscar los

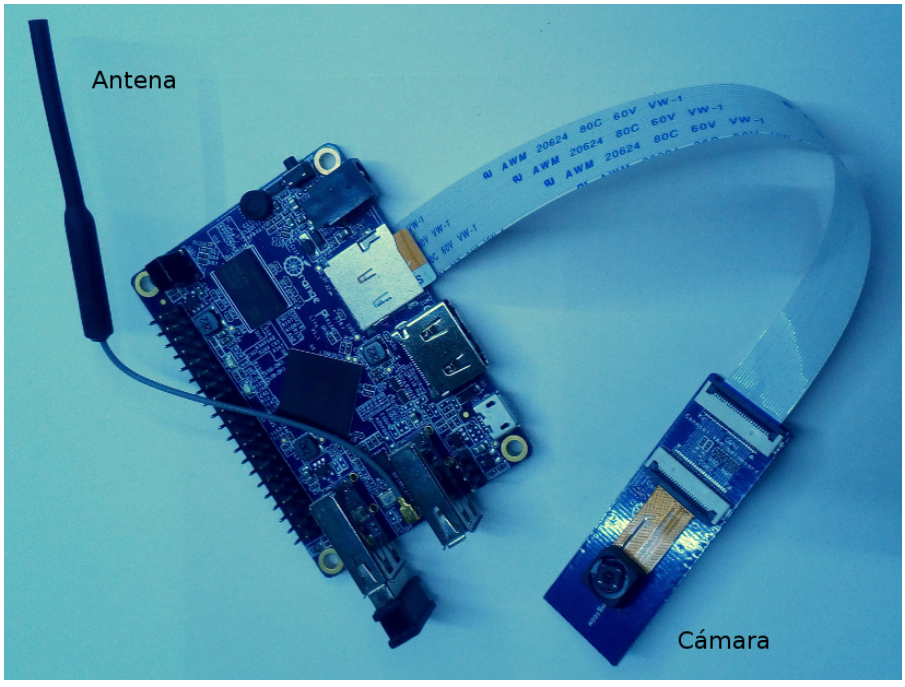

Figura 2. Detalle de la plataforma H3 Quad-core Cortex-A7 utilizada en el módulo bróker

Fuente: elaboración propia.

círculos rojos colocados en el cuerpo de los otros dos robots. Si no los ve luego de un proceso de exploración girando sobre su eje, se debe desplazar en el espacio libre de forma aleatoria un par de segundos, y explorar de nuevo. Si localiza el *landmark*, debe navegar hacia él.

Estos algoritmos de comportamiento definen la funcionalidad del sistema, y están diseñados para que los robots se agrupen en alguna parte del ambiente. El punto de agrupación en esta tarea puede condicionarse a características específicas del ambiente. Por ejemplo, que los robots solo se detengan si el nivel de luz es alto (en un área de carga por paneles solares, por ejemplo), pero si está por debajo de algún umbral, que continúen la navegación aleatoria. Este comportamiento de agrupación puede hacer que los robots se agrupen en ciertas áreas luego de desarrollar alguna tarea (limpieza, búsqueda, ...).

Se realizaron cerca de 50 experimentos para diferentes tamaños y configuraciones del ambiente de navegación. El ambiente de navegación se mantuvo siempre con forma cuadrada, pero su tamaño se varió entre 30 cm × 30 cm y 2,4 m × 2,4 m. A fin de analizar la independencia de comportamiento con respecto a la arquitectura OpenRRArch, se desarrolló un modelo ideal de comportamiento considerando los tiempos del prototipo real (retardo de motores, sensores y tiempos de espera asignados en los algoritmos), para su simulación por computador. La simulación se realizó en Python mediante Pygame. Los resultados se resumen en la figura 3.

Estos resultados muestran coherencia en los tiempos promedios de convergencia sin importar si se trata de plataforma real o simulación (estadísticamente se puede demostrar que son iguales). También se observa que las variaciones en el tiempo dependen del tamaño del ambiente, y que de hecho hay una relación de proporcionalidad. Estos resultados validan las hipótesis de desempeño de la arquitectura.

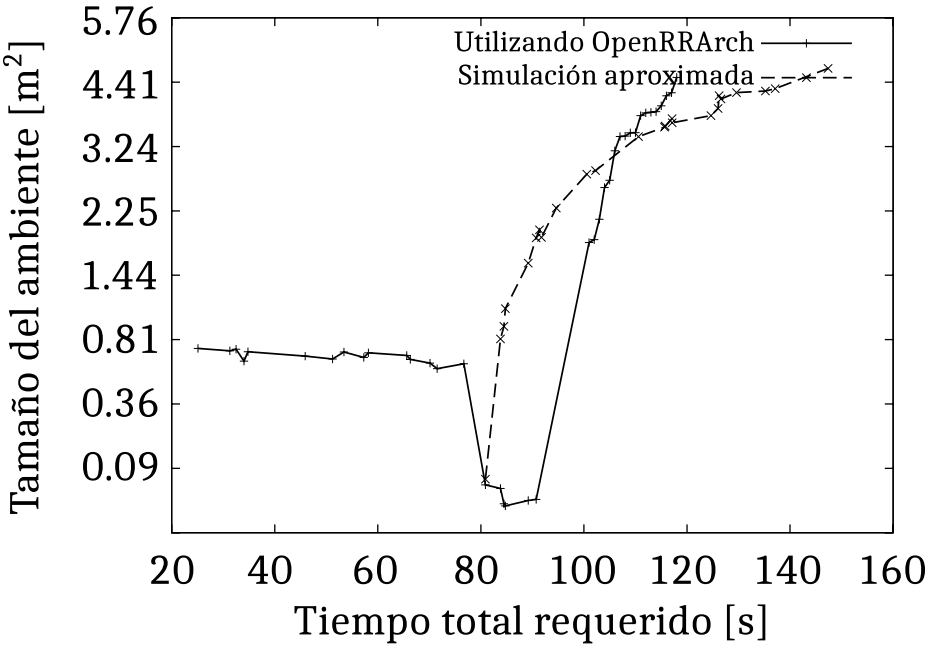

Figura 3. Resultados de tiempo de convergencia del algoritmo para diferentes tamaños de ambiente frente a simulación del caso ideal

Fuente: elaboración propia.

CONCLUSIONES

El artículo propone una arquitectura distribuida para la implementación de sistemas multiagente denominada OpenRRArch. Dicha arquitectura se pretende utilizar para analizar el desempeño de esquemas de navegación de pequeños robots autónomos con control reactivo. Como criterios de diseño se considera el uso de herramientas *hardware* y *software open source*, así como características de robustez, escalabilidad y confiabilidad. OpenRRAch permite la rápida implementación, a bajo costo, y con alto desempeño, de sistemas con capacidad de cooperación y comunicación en tiempo real. La arquitectura permite la integración de agentes con diferentes capacidades de procesamiento, lo que permite adaptación a las necesidades de la tarea. Las funciones de cada uno de los agentes pueden reducirse o aumentarse a partir de su código, o seleccionando otro *hardware* embebido. Mediante pruebas de laboratorio se pudo determinar que para una básica tarea de agrupamiento la arquitectura no altera el funcionamiento del algoritmo.

REFERENCIAS BIBLIOGRÁFICAS

Carlson, D. y Schrader, A. (2012). Dynamix: An open plug-and-play context framework for android. En *3rd International Conference on the Internet of Things (IOT 2012)* (pp. 1-8).

Contreras, J.; Martínez, F. y Martínez, F. (2015). Path planning for mobile robots based on visibility graphs and A* algorithm. *Proceedings of Spie*, 9631, 1-5.

Elwahab, A.; Nouali, O.; Moussaoui, S. y Derder, A. (2015). A BLE-based data collection system for iot. En *First International Conference on New Technologies of Information and Communication* (NTIC 2015) (pp. 1-5).

Espinosa, O.; Castañeda, L. y Martínez, F. (2015). Minimalist artificial eye for autonomous robots and path planning. Lecture Notes In Computer Science. *Intelligent Data Engineering and Automated Learning*, 9375, 232-238.

Jacinto, E.; Giral, M. y Martínez, F. (2016). Modelo de navegación colectiva multiagente basado en el quorum sensing bacterial. *Revista Tecnura, 20*(47), 29-38.

Kiwoong, K.; Minkeun, H.; Taehong, K.; Seong, H. y Daeyoung, K. (2012). The stateless point to point routing protocol based on shortcut tree routing algorithm for IP-WSN. En 3rd *International Conference on the Internet of Things* (IOT 2012) (pp. 1-8).

Kouicem, A.; Chibani, A.; Tari, A.; Amirat, Y. y Tari, Z. (2014). Dynamic services selection approach for the composition of complex services in the web of objects. En *IEEE World Forum on Internet of Things (WF-IOT 2014)* (pp. 298-303).

Lee, H.; Cho, Y. y Lee, B. (2013). Robot-controlled fusion. *Electronics Letters, 49*(15), 912-912.

Luna, E.; Silva, A. y Costa, C. (2016). An attentional model for autonomous mobile robots. *IEEE Systems Journal*, 99, 1-12.

Martínez, F., y Delgado, A. (2010). Hardware emulation of bacterial quorum sensing. *Advanced Intelligent Computing Theories and Applications, Lecture Notes in Computer Science*, 6215, 329-336.

Martínez, F.; Jacinto, E. y Martínez, F. (2014). Using the delaunay triangulation and voronoi diagrams for navigation in observable environments. *Revista Tecnura*, 18, 81-87.

Montiel, H.; Valderrama, H. y Martínez, F. (2016). Using embedded robotic platform and problem-based learning for engineering education. *Smart Innovation, Systems and Technologies*, 59, 435-445.

Oral, T. y Polat, F. (2016). MOD* Lite: an incremental path planning algorithm taking care of multiple objectives. *IEEE Transactions on Cybernetics, 46*(1), 245-257.

Sirin, E.; Parsia, B. y Hendler, J. (2005). Template-based composition of semantic web services. En *AAAI Fall Symposium on Agents and the Semantic Web* (pp. 85-92).

Tari, K.; Amirat, Y.; Chibani, A.; Yachir, A. y Mellouk, A. (2010). Context-aware dynamic service

composition in ubiquitous environment. En *IEEE International Conference on Communications (ICC 2010)* (p. 1-6).

Teatro, T.; Eklund, M. y Milman, R. (2014). Nonlinear model predictive control for omnidirectional robot motion planning and tracking with avoidance of moving obstacles. *Canadian Journal of Electrical and Computer Engineering, 37*(3), 151-156.

Wenfeng, L.; Junrong, B. y Weiming, S. (2011). Collaborative wireless sensor networks: A survey. En *IEEE International Conference on Systems, Man, and Cybernetics (SMC 2011)* (pp. 2614-2619).

Yachir, A.; Amirat, Y.; Chibani, A. y Badache, N. (2012). Towards an event-aware approach for ubiquitous computing based on automatic service composition and selection. *Annales des Télécommunications, 67*(7), 341-353.