https://doi.org/10.17979/spudc.9788497498081.0935

ARQUITECTURA DE COMUNICACIONES DE TIEMPO REAL PARA ROBOTS MODULARES BASADOS EN ROS

Eduardo Munera, Jose-Luis Poza-Luján, Juan-Luis Posadas-Yagüe,
J.Francisco Blanes Noguera and Jose E. Simo

Instituto de Automática e Informática Industrial (ai2) Universitat Politècnica de València (UPV)
Cami de vera, s/n 46022 Valencia, España. e-mail: (emunera, jopolu, jposadas, pblanes, jsimo)@ai2.upv.es

Resumen

Actualmente, las plataformas para robots que implementan ROS son ampliamente usadas en investigación. Éstas han probado ofrecer una solución fiable para reutilizar algoritmos. No obstante, estos cuentan con ciertos inconvenientes. Aunque la arquitectura ROS ha sido diseñada con la posibilidad de trabajar de forma distribuida, este diseño presenta algunos problemas al gestionar un gran numero de dispositivos los cuales requieren de un flujo de datos en tiempo real. En este trabajo se presenta una arquitectura de comunicaciones para solucionar este problema. A demás se valida dicha arquitectura por medio de un robot modular a partir de dispositivos totalmente desacoplados. Un robot modular puede presentarse como un Sistema de Control Distribuido (DCS, por sus iniciales en inglés), donde las tasas de transmisión y el flujo de datos debe ser acotado para poder garantizar una correcta ejecución. Por lo tanto, se detalla como solucionar el problema de comunicación distribuida entre nodos ROS posibilita el intercambio de datos de forma fácil y segura. Como consecuencia se propicia el desarrollo de sistemas modulares basados en ROS. Por ultimo se realiza una comparación entre el desempeño del sistema obtenido cuando se implementa una distribución ROS clásica, y el obtenido cuando se utiliza la arquitectura de comunicaciones propuesta.

Palabras clave: Palabras clave: Sistemas Distribuidos de Control, Arquitecturas de comunicación, Robots Modulares, ROS

1. INTRODUCCIÓN

El desarrollo y aplicación de robots modulares ofrecen ventajas tales como la re-utilización de los módulos, el procesamiento descentralizado o la abstracción del bajo nivel. No obstante, al igual que cualquier otro Sistema de Control Distribuido (DCS), deben cumplirse unos requisitos de tiempo real en las comunicaciones [10]. El uso de sistemas modulares basados en ROS ofrece amplias ventajas como la estandarización de módulos, la reutilización de los algoritmos o la opimización de la ejecución de los mismos [16]. A pesar de ello, dados los requisitos de los DCS, de entre lo middlewares de robots modulares, ROS ofrece algunos claros inconvenientes para el desarrollo de robots modulares distribuidossobre una red. Como se comentará a continuación, el gran numero de dispositivos conectados a la red junto con la necesidad de exportar el núcleo (ROS Master) desde un dispositivo maestro previamente conocido, dificultan el desarrollo de este tipo de sistemas.

En una red distribuida ROS el dispositivo encargado de la ejecución del ROS Master (nodo ROS maestro) se encarga de gestionar las comunicaciones y resolver el espacio de nombres. De tal forma, el primer problema consiste en la necesidad de conocer de forma anticipada la dirección IP y el puerto en el cual se encuentra configurado el ROS Master con el fin de aceptar conexiones de otros dispositivos. Todos los dispositivos que formen parte del sistema distribuido y deseen implementar ROS deben ser configurados de esta forma para contactar con este dispositivo maestro. En caso de cambiar el nodo maestro o algunos de estos parámetros, todos ellos deberán volver a ser configurados. La distribución del nodo ROS maestro también impide la integración de diversas redes ya que dicho nodo debe ser directamente accesible.

El siguiente problema es la dependencia que vincula a cada uno de los dispositivos con el nodo ROS maestro. Teniendo ésto en cuenta, los dispositivos no pueden trabajar de forma independiente, siendo incapaces de ejecutar los módulos ROS si el ROS Master distribuido no esta en marcha o no es accesible. Además de ello, los diversos dispositivos no serán conscientes de su existencia mutua en la red sin llevar a cabo una comunicación iniciada por el ROS Master. A partir de lo expuesto anteriormente, se plantean los siguientes objetivos:

- Diseñar cada dispositivo distribuido ROS como su propio nodo ROS maestro, evitando dependencias de sistemas externos para la ejecución de los diversos procesos ROS.
- Generar una conexión Multi-punto entre todos los dispositivos ROS conectados en la red para el intercambio de información. Esta im-

plementación incluirá mecanismos de mejora tales como el descubrimiento automático de dispositivos y servicios de proxy para conectar redes internas.

- Estudiar el desempeño de la arquitectura de comunicación propuesta como parte de un DCS de tiempo real, caracterizado como un robot modular.

De acuerdo con estos objetivos el articulo se estructura de la siguiente manera: En primer lugar la Sección 2 presenta trabajos relacionados con la aplicación de ROS en sistemas distribuidos mediante el uso de dispositivos ROS Ready. A continuación, la Sección 3 introduce el concepto de Smart Resource con el fin de proporcionar un soporte de ejecución para el desarrollo de robots modulares compatibles con ROS basados en servicios. La arquitectura de comunicación necesaria para establecer el flujo de datos en tiempo real es detallada a lo largo de la Sección 4. En la Sección 5 se presentan y analizan los experimentos y resultados relacionados con el desempeño del sistema. Finalmente, en la Sección 6 se detallaran las conclusiones y se presentara el trabajo futuro.

2. TRABAJO RELACIONADO

Con el objetivo de distribuir los mensajes de sensores, actuadores y controladores entre los diversos componentes de un nodo, es necesario ofrecer un sistema de comunicaciones fiable, así como de procesos implicados en la operación del robot. De la misma forma que un ordenador necesita de un sistema operativo, el uso de un sistema análogo enfocado a los elementos distribuidos de un robot ofrece un gran numero de ventajas [11].

De todas las opciones disponibles el sistema ROS[16] ha demostrado ser el mas extendido ya que en los últimos años se ha producido un incremento considerable del numero de desarrollos basados en ROS [5]. La arquitectura de ros se basa en bloques de ejecución denominados como nodos conectados mediante un sistema de comunicaciones de tipo publicación/suscripción. ROS es también una plataforma de software abierto, por lo que es altamente modificable y adaptable a las necesidades especificas independientemente del tipo de robot o su configuración.

Cada uno de los dispositivos que implementan ROS pasan a convertirse en un proveedor de servicios distribuidos de control [17] definidos como dispositivos ROS-Ready [2]. el hecho de proporcionar servicios de control compatibles con ROS, ofrece una abstracción de alto nivel para el usuario. Entre las principales ventajas de esta abstracción se encuentra la posibilidad de reutilizar dispositivos entre plataformas o simular parte del sistema mediante la virtualización de ciertos componentes. A fin de ofrecer un ejemplo sobre este tipo de integración, en [3] se presenta un robot que utiliza ROS para integrar todos los sensores de abordo. ASTRO [19] ofrece una arquitectura orientada a servicios paraa gestionar robots en escenarios complejos.

Además, teniendo en cuenta la tendencia actual, es muy importante la distribución de la información de control entre los dispositivos implicados en la red de control a fin de ofrecer una cierta funcionalidad a los clientes. Esta distribución se realiza por medio de middlewares que hacen de intermediarios entre los dispositivos y la red [1]. Existe una amplia variedad de middlewares orientados a la robótica que ofrecen mecanismos de comunicación [7]. Las capacidades de comunicación implementadas por ROS también ofrecen la posibilidad de generar una distribución de las comunicaciones entre dispositivos, por lo que ROS puede ser considerado potencialmente un middleware al que se deben añadir una serie de componentes para ser empleado en sistemas de robots distribuidos.

También se pueden encontrar trabajos en los cuales se implementa un paradigma DDS (Data Distribution Service)[15] de tiempo real para proveer una capa de publicación/suscripción vinculada a la ofrecida por ROS, un claro ejemplo de ello es el trabajo introducido en [18]. Mediante este tipo de técnicas puede ocultarse el sistema de comunicaciones a fin de ofrecer las funcionalidades de ROS . En otros casos se proponen la utilización de soluciones ad-hoc tal y como las que se presentan en [9] mediante conexiones TCP/IP o en [6] mediante websockets.

En los casos descritos anteriormente, los servicios ofrecidos por ROS pueden ser comunicarse gracias a la traducción de mensajes. Este articulo ofrece una arquitectura de comunicación en la que los dispositivos definidos como Smart Resources pasan a convertirse en dispositivos ROS-Ready interconectados en una red como parte de un DCS.

3. TRABAJO PREVIO: SMART RESOURCES

La principal finalidad de ofrecer las funcionalidades implementadas en un servicio como servicios consiste en abstraer al cliente de dichos servicios de las tecnologías usadas, como por ejemplo el acceso el acceso de bajo nivel. En tal caso los dispositivos dejan de ser considerados como sistemas distribuidos para gestionarse como recursos distribuidos. En este articulo se hace uso de los recursos

distribuidos implementados como Smart Resources tal y como se presentan en [14]. Los Smart Resources se presentan como una solución adaptable a cualquier tipo de aplicación gracias a sus capacidades de reconfiguración dinámica [12] y de abstracción. Por todo lo anterior, sus principales características y su integración en un sistema robótico se detallarán a lo largo de esta sección.

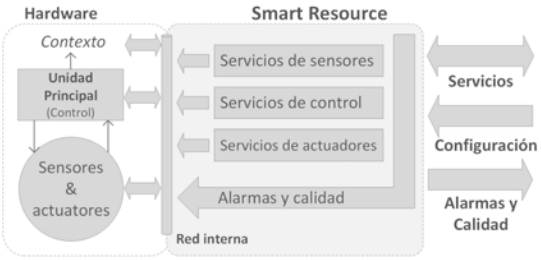

Figura 1: Esquema de diseño de un Smart Resource orientado a aplicaciones robóticas

3.1. INTEGRACIÓN EN PLATAFORMAS ROBOTICAS

Los servicios proporcionados por los Smart Resources pueden ser requeridos por cualquier tipo de robot o dispositivo para la implementación de DCSs. Por lo tanto los servicios proporcionados usualmente se vincularán a tareas de sensorización, actuación y control. Estos servicios pueden ser proporcionados por diferentes plataformas hardware (físicas) o simuladas (virtuales) las cuales ofrecen un interfaz de servicio común.

La distribución de servicios en los Smart Resources se realiza a través del sistema de comunicaciones. En el caso del sistema presentado en el artículo, se propone un sistema específico. Este sistema es caracterizado como un middleware de comunicaciones cuyas principales funciones son: detección de dispositivos en la red, gestión de nodos y flujo de datos basado en un sistema de publicación/suscripción [8].

Al igual que otras implementaciones de publicación/suscripción la comunicación esta dirigida a topics, siendo éstos definidos como el flujo de datos asociados a una etiqueta identificadora. El sistema propuesto ofrece topics relacionados con la configuración de los Smart Resources, los servicios y las calidades asociadas a la provisión de dichos servicios (QoS).

Como ha sido mencionado anteriormente, los Smart Resources ofrecen la capacidad de adaptar el sistema para funcionar dentro de unos márgenes de calidad. Cuando se proporciona un servicio, éste debe ser configurado para cumplir las necesidades del cliente dentro de dichos márgenes. La mayor parte de estos requisitos de calidad se configuran en función de unas restricciones que pueden ser temporales , espaciales, de fiabilidad o de desempeño de las tareas.

La adaptación del Smart Resource se realiza a través de perfiles de sistema. Un "perfil de sistema"se define como una configuración del sistema en la que se debe operar con unas restricciones concretas. Los perfiles del sistema son programados previamente en el Smart Resource. Durante la ejecución, las medidas de calidad evaluaran el perfil de sistema activo a fin de comprobar si debe producirse alguna adaptación debida al incumplimiento de los requisitos del servicio. Dicho incumplimiento será notificado por un sistema de alarmas que desencadenarán el evento de adaptación. Si las calidades del sistema no se ajustan a ninguno de los perfiles disponibles, se ejecutará aquel que ofrezca una mejor aproximación. Estos eventos no adaptables deben considerarse ocasionales antes una situación extrema ya que, de lo contrario, implicará un mal diseño de los perfiles del sistema basándose en unos requisitos no realistas por parte del cliente del servicio.

Aunque estos mecanismos de adaptación ofrecen la capacidad de optimizar el funcionamiento de de los servicios del Smart Resource, la adaptación al contexto también supone un elemento crítico para la eficiencia del sistema. Durante el desempeño de sus funciones un robot puede encontrarse con diversos contextos o entornos dinámicos. De acuerdo con ésto, los servicios deberán ser adaptados al contexto en el cual se desenvuelve el robot. Los Smart Resource pueden ser configurados para manejar diferentes servicios y perfiles de sistema de acuerdo con las necesidades del robot.

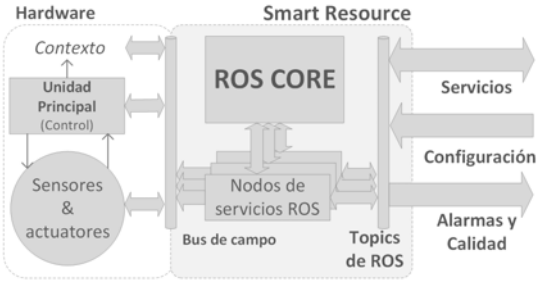

Figura 2: Diseño de un Smart Resource basado en ROS

3.2. INTEGRACIÓN EN ENTORNOS ROS

A pesar de que los Smart Resources han sido originalmente diseñados para trabajar con su propio middleware de ejecución, su diseño ha sido modificado para basar su ejecución en el núcleo de

ROS [13]. Para ello, cada Smart Resource debe ejecutar su propia instancia del ROS Master con el fin de gestionar los nodos ROS requeridos, y de esta forma ofrecer el servicio requerido. De la misma manera, la red de comunicaciones interna ha sido remplazada por el sistema de publicación/suscripción ofrecido por ROS. Los detalles de esta implementación pueden apreciarse en la Figura **??**.

4. ARQUITECTURA ROS MULTI-PEER

Una vez definido el marco de trabajo, se detallará la arquitectura propuesta a lo largo de esta sección. Dado que el objetivo final de esta propuesta es ofrecer un solución distribuida y autónoma para el desarrollo de robots modulares, el flujo de datos conformará una etapa critica del proceso. A continuación se detalla el proceso para la implementación de la arquitectura para comunicar un sistema distribuido entre iguales (multi-peer) formado por Smart Resources basados en ROS. Esta arquitectura, llamada RMPA (ROS Multi-Peer Architecture) ofrece una implementación completa desde el nivel de comunicación, el establecimiento de la red multi-peer, u otros mecanismos mas sofisticados como la detección automática de dispositivos en la red y el interfaz a la red de comunicaciones local de ROS.

4.1. NIVEL DE SMART RESOURCE

Los Smart Resources han sido diseñados como módulos independientes cuya finalidad esta orientada a interactuar con otros módulos mediante el ofrecimiento de servicios (ya sean otros Smart Resources o cualquier otro tipo de dispositivo).

4.2. TOPOLOGIA MULTI-PEER

Cada módulo es caracterizado como un peer dentro de la topología de la red. Cada peer en el sistema está conectado con el resto formando un mandala como el descrito en la Figura 3.

Los peers de la red definen sus conexiones utilizando un GUID (Global Unique IDentifier) definido como un numero de 64 bits el cual se establece al combinar la dirección IP (32 bits sin signo) y un puerto de conexión (16 bits). Este GUID no se usa solo como identificador unitario sino que también establece el orden entre los peers de la red. Este orden es utilizado para determinar la gestión de los mensajes en la secuencia de comunicación.

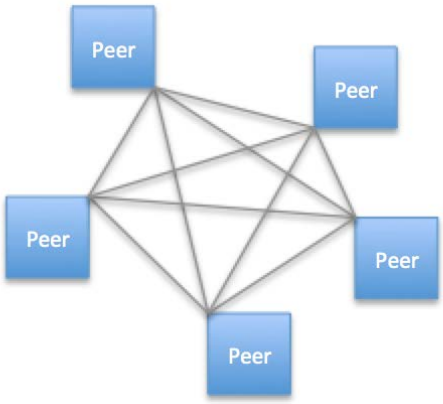

Figura 3: Topología Multi-peer

4.3. NIVEL DE COMUNICACIÓN

La comunicación entre los peers se establece a través de una conexión TCP/IP entre nodos. Para ello cada nodo necesita de un hilo servidor para aceptar la conexión de otros peers. Dado un peer con un GUID específico, éste deberá iniciar la conexión con los peers que dispongan de un GUID superior. Por el contrario, este peer esperará que los peers con un GUID inferior sean los encargados de iniciar la comunicación con el. El proceso de conexión se encuentra detallado en la Figura 4.

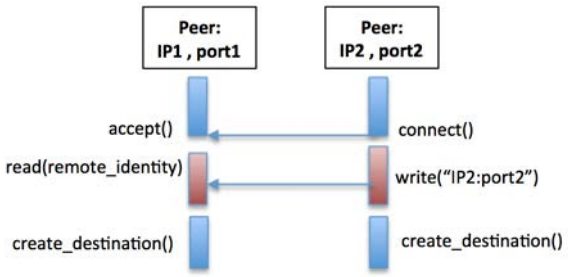

Figura 4: Conexión entre peers

4.4. MECANISMO DE DESCUBRIMIENTO

Una vez que un nodo (peer) se conecta a la red, se establece un canal de difusión múltiple (multicast) mediante la asignación de una dirección IP y de un puerto a través del cual atenderán a las conexiones. Cada peer notificará su GUID a través del canal de difusión. Los peers también notificarán su propio GUID cada vez que se reciba la difusión de un nuevo GUID. Cuando todos los GUID han difundido en la red su GUID, el canal se mantiene inactivo. No obstante cada cierto tiempo, establecido por defecto en 5 segundos, se comprobará que

la red se mantiene activa mediante un mensaje de *heartbeat*.

4.5. INTERFAZ A LAS COMUNICACIONES ROS

Tal y como se ha mencionado anteriormente, el sistema de distribución de ROS ofrece algunos problemas relacionados con la dependencia de distribución del ROS Master a partir del dispositivo maestro. Por ello el principal objetivo del interfaz RMPA es migrar de un sistema distribuido con un ROS Master compartido (tal y como se describe en la Figura 5) a un sistema distribuido formado por dispositivos independientes que ejecutan su propio ROS Master local y que intercambian información a través de la red Multi-peer (como se muestra en la Figura 6).

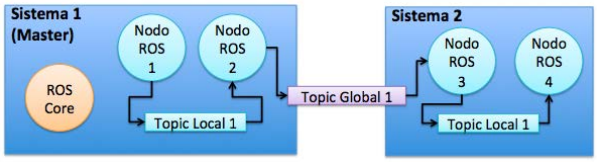

Figura 5: Distribución clásica de ROS

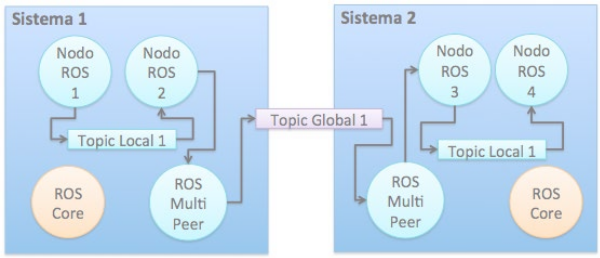

Figura 6: Distribución de ROS basada en Multi-peer

5. EXPERIMENTOS Y RESULTADOS

A lo largo de esta sección se presentan un conjunto de experimentos realizados, y los resultados obtenidos, con el fin de evaluar de la arquitectura propuesta. Para ello se ha diseñado un caso de estudio en el cual se emplean diferentes tipos de dispositivos, los cuales están conectados a una misma red y realizan diversas funciones dentro de la cadena de control.

La Figura 7 detalla el entorno experimental al completo, formado por 3 placas BeagleBone Black (BBB) [4] y un ordenador (PC). El objetivo de este entorno es disponer de los tres elementos básicos de una cadena de control:

- Un elementos generador del que se obtenga información. Este elemento bien puede ser un emisor de una orden como la la lectura de un sensor.
- Un elemento de control que procesará la información generada por el elemento anterior y que generará una acción de control.
- Un elemento actuador que recibirá la acción de control y actuará en consecuencia.

El elemento generador de información, publicará la información en un *topic* al que estará suscrito el elemento de control. A su vez, el resultado de la acción de control se publicará en otro *topic* al que estará suscrito el elemento actuador. Además, se deberá disponer de un elemento monitor de los elementos anteriores, para comprobar los tiempos en función de si se está experimentando una configuración ROS clásica o una configuración RMPA.

Tal y como se puede comprobar, la primera placa BBB se encarga de leer comandos de un joystick USB para posteriormente publicar dicha información en el topic *Command Information*. La segunda BBB lee esta información a fin de calcular una referencia de velocidad que será publicada en el topic *Velocity Order*. La última placa se comunicará con el robot para realizar el desplazamiento requerido. Mientras, el PC conectado a la red hará las funciones de monitor, registrando el flujo de datos y comprobando que las condiciones de tiempo real propuestas han sido cumplidas.

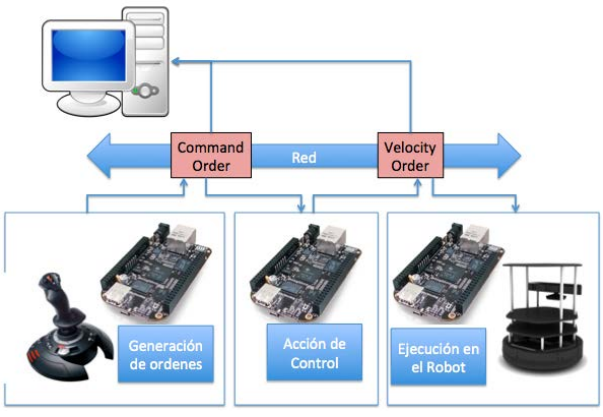

Figura 7: Configuración experimental

De acuerdo con esta configuración se han desarrollado dos tipos de experimentos. El primero de ellos se basa en usar una distribución de datos clásica de ROS en la cual el ROS Master es compartido entre todos los dispositivos de la red, tal y como se planteaba en la Figura 5. En este caso el ROS Core será ejecutado en el PC, mientras que las tres placas BBB han sido configuradas con la

dirección IP, y el puerto necesario para su exportación. En el segundo experimento, cada dispositivo ha sido implementado como un Smart Resource ROS el cual ejecuta su propio ROS Master de forma local y establecerán un flujo de datos a través de la comunicación RMPA.

A lo largo de ambos experimentos se almacenaron los tiempos de distribución de los datos para un posterior análisis de las tasas de transferencia y cálculo del retardo derivado del cómputo de la referencia de velocidad. La tasa de recepción del topic *Command Information* pasa a ser expresada como textitCMD, mientras que *VEL* pasará a expresar la del topic *Velocity Order*.La frecuencia de publicación de ambos topics se establece en 10Hz.

Tal y como se observa en las Figuras 9 y 8, ambas tasas de recepción (CMD y VEL) se agrupan entorno a 0.1s según lo previsto. Analizando el tiempo de cálculo de la referencia de velocidad puede apreciarse como es considerablemente reducido en caso de implementar la RMPA. Como principal inconveniente de esta implementación, se puede apreciar como la dispersión de las tasas de recepción sufre un ligero incremento en relación a los obtenidas con la distribución clásica de ROS:

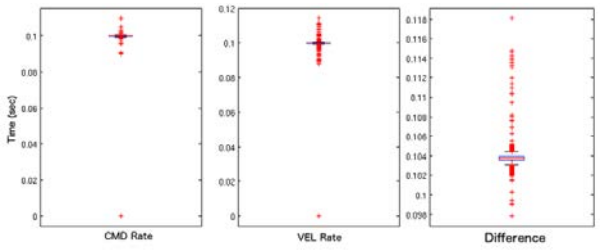

Figura 8: Tasas de distribución y cómputo empleando la distribución clásica de ROS

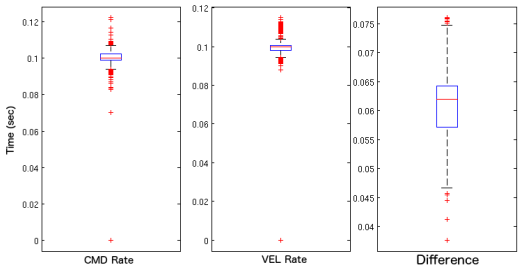

Figura 9: Tasas de distribución y cómputo empleando la distribución mediante RMPA

En la Tabla 1 se ofrece una comparación numérica entre ambos experimentos. Las tasas de distribución se mantienen coherentes a los 10Hz configurados. En el caso del cómputo de la velocidad se puede comprobar cómo el tiempo se ve reducido a prácticamente la mitad. Este efecto se produce como consecuencia de la ejecución de un ROS Master local que no debe ser exportado desde un dispositivo maestro.

Cuadro 1: Comparación entre las distribuciones RMPA y ROS

	Tasa Recepción RMPA	Media (seg) ROS
Command Order	0.0999	0.0999
Velocity Order	0.0999	0.0999
Tiempo de Calculo Velocidad	0.0613	0.1038

6. CONCLUSIONES Y TRABAJO FUTURO

Como conclusión principal debe destacarse que los objetivos iniciales han sido cumplidos gracias a la adaptación de la topología de los Smart Resources dentro del entorno ROS. De esta forma, se incrementa la capacidad de implementación de dispositivos distribuidos y autónomos con menor latencia en mensajes.

Mediante el desarrollo de una arquitectura multipeer para ROS se elimina la dependencia entre dispositivos de exportar el ROS Core, configurándose cada uno como su propio maestro. Además, se proporcionan mejoras adicionales en el sistema de comunicación tales como el descubrimiento de peers en la red y la optimización de los mensajes que se envían a través de la red. Además, en el caso de una configuración ROS clásica, cada elemento debe ejecutar todo el código de control correspondiente a su nodo, mientras que en la configuración propuesta RMPA, el código de control es fácil de distribuir, dado que se extienden los topics de ROS de un elemento a todos los elementos del sistema. Esto es lo que puede propiciar poder reutilizar el código de un elemento en otro elemento.

Finalmente, esta arquitectura ha sido probada mediante la ejecución de diversas pruebas en las cuales diversos Smart Resources que implementan ROS trabajan en un contexto distribuido de control. Estos resultados han sido comparados con el desempeño obtenido con una distribución clásica de ROS. Como resultado, se ha demostrado que en ambos casos se cumplen las tasas de distribución, no obstante en el caso de RMPA se prueba como el hecho de disponer de un ROS Master local, se produce un incremento en la eficiencia de la ejecución.

Como trabajo futuro, se plantea analizar cómo la calidad de la información y las capacidades de configuración de los Smart Resources pueden mejorar por medio del uso de la arquitectura RM-PA. Además, se debe estudiar cómo la inclusión de meta-datos en los mensajes y de información persistente en el sistema puede propiciar una mejora de la eficiencia global del mismo.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación de España (MICINN): CICYT project M2C2 Co-diseño de sistemas de control con criticidad mixta basado en misiones TIN2014-56158-C4-4-P y el programa PAID (Universidad Politécnica de Valencia): UPV-PAID-FPI-2013.

Referencias

[1] Jameela Al-Jaroodi and Nader Mohamed. Service-oriented middleware: a survey. *Journal of Network and Computer Applications*, 35(1):211–220, 2012.

[2] André Araújo, David Portugal, Micael S Couceiro, Jorge Sales, and Rui P Rocha. Development of a compact mobile robot integrated in ros middleware. *Revista Iberoamericana de Automática e Informática Industrial RIAI*, 11(3):315–326, 2014.

[3] Heping Chen, Hongtai Cheng, Biao Zhang, Jianjun Wang, Thomas Fuhlbrigge, and Jian Liu. Semiautonomous industrial mobile manipulation for industrial applications. In *Cyber Technology in Automation, Control and Intelligent Systems (CYBER), 2013 IEEE 3rd Annual International Conference on*, pages 361–366. IEEE, 2013.

[4] Gerald Coley. Beaglebone black system reference manual. *Texas Instruments, Dallas*, 2013.

[5] Steve Cousins. Exponential growth of ros [ros topics]. *IEEE Robotics & Automation Magazine*, 1(18):19–20, 2011.

[6] Christopher Crick, Graylin Jay, Sarah Osentoski, Benjamin Pitzer, and Odest Chadwicke Jenkins. Rosbridge: Ros for non-ros users. In *Proceedings of the 15th International Symposium on Robotics Research*, 2011.

[7] Ayssam Elkady and Tarek Sobh. Robotics middleware: A comprehensive literature survey and attribute-based bibliography. *Journal of Robotics*, 2012, 2012.

[8] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, 2003.

[9] Jeong Seok Kang, Dong Uk Yu, and Hong Seong Park. A robot software bridge for interconnecting opros with ros. In *Ubiquitous Robots and Ambient Intelligence (URAI), 2012 9th International Conference on*, pages 296–297. IEEE, 2012.

[10] Feng-Li Lian, James Moyne, and Dawn Tilbury. Network design consideration for distributed control systems. *IEEE Transactions on Control Systems Technology*, 10(2):297–307, 2002.

[11] Fan Liu, Ajit Narayanan, and Quan Bai. Real-time systems. 2000.

[12] Eduardo Munera, Jose-Luis Poza-Lujan, Juan-Luis Posadas-Yagüe, Manuel Muñoz, and Juan Fco. Blanes Noguera. Poster: Context-aware adaptation mechanism for smart resources. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, SenSys '15, pages 431–432, New York, NY, USA, 2015. ACM.

[13] Eduardo Munera, Jose-Luis Poza-Lujan, Juan-Luis Posadas-Yagüe, Jose-Enrique Simó-Ten, and Francisco Blanes. Smart resource integration on ros-based systems: Highly decoupled resources for a modular and scalable robot development. In *Distributed Computing and Artificial Intelligence, 13th International Conference*, pages 331–338. Springer, 2016.

[14] Eduardo Munera, Jose-Luis Poza-Lujan, Juan-Luis Posadas-Yagüe, José-Enrique Simó-Ten, and Juan Fco Blanes Noguera. Dynamic reconfiguration of a rgbd sensor based on qos and qoc requirements in distributed systems. *Sensors*, 15(8):18080–18101, 2015.

[15] Gerardo Pardo-Castellote. Omg data-distribution service: Architectural overview. In *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, pages 200–206. IEEE, 2003.

[16] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[17] Sekou L Remy and M Brian Blake. Distributed service-oriented robotics. *Internet Computing, IEEE*, 15(2):70–74, 2011.

[18] Yamnia Rodríguez, Carlos Alejo, Irene Alejo, and Antidio Viguria Jiménez. Animo, framework to simplify the real-time distributed communication. In *UBICITEC*, pages 16–26. Citeseer, 2014.

[19] Jacques Saraydaryan, Fabrice Jumel, and Adrien Guenard. Astro: Architecture of services toward robotic objects. *International Journal of Computer Science Issues (IJCSI)*, 11(4):1, 2014.