ISSN: 1692-7257
Revista Colombiana de Tecnologías de Avanzada

# ROS SISTEMA OPERATIVO PARA ROBOTICA, NOCIONES Y APLICACIONES

**NUÑEZ TORRES Manuel J[1], LEÓN CARLOS Fredy H[1], CARDENAS Pedro F[2].**

**Universidad Nacional de Colombia-Sede Bogotá**
**Departamento de Ingeniería Mecánica y mecatrónica (DIMM)**
Ciudad Universitaria. Bogotá DC, Colombia.
Tel.: 57-1-3165000, Ext. 11106
E-mail: {mjnunezt, fhleonc}@unal.edu.co.

**Abstract:** In this paper a brief introduction to operating systems for robots, the concept of ROS, philosophy which it pursued, the facilities and the current perspective in different fields. Besides a short introduction to the method of operation and necessary elements that compose it. Continue, Implementing a basic example simulating a sensor on this platform, and also, a practical application of many tools that ROS provides in the integration of serial manipulator 5 DOF to the robots supporting for this system.

**Keywords:** MRS, Node, ROS, Sensor, Serial Robot.

**Resumen:** En el presente trabajo se hace una breve introducción a los sistemas operativos para robots, el concepto de ROS, la filosofía que persigue, las facilidades que ofrece y la perspectiva actual en diferentes ámbitos. Además, se hace una pequeña introducción a la forma de funcionamiento y necesariamente a los elementos que la componen. Se expone un ejemplo básico implementando la simulación de un sensor en esta plataforma, y también, una aplicación práctica de muchas de las herramientas que proporciona ROS en la integración de un manipulador serial 5 DGL a la serie de robots que ya soporta este sistema.

**Palabras clave:** MRS, nodos, robot serial, ROS, sensor.

## 1. INTRODUCION

Actualmente la implementación de los sistemas robóticos se hace más compleja, en gran parte, debido a que las aplicaciones requieren un mayor grado de abstracción e innovación. El trabajo para el desarrollo de estos sistemas, en particular el software, debido al grado multidisciplinar requerido en los grupos de desarrollo, resulta ser en ocasiones desbordante y e irrealizable; por lo tanto desarrollar un aplicación en robótica desde cero, es un trabajo que para las universidades y la industria resulta ser una barrera que impide un alto nivel de apropiación.

En la investigación académica se percibe que múltiples grupos trabajan en la solución al mismo problema, desarrollan herramientas similares que son poco difundidas, y cuando varias de estas herramientas se intentan usar integradas, resulta casi siempre arduo, pues no se rigen bajo un mismo esquema de desarrollo. Inclusive, luego de finalizar las investigaciones, muchas de estas herramientas y aplicaciones se desactualizan rápidamente, debido a que no hay quien continúe con la investigación o quien la aplique en una necesidad del día a día.

En el caso de la industria, las herramientas desarrolladas son limitadas y muy cerradas -solo para ciertos equipos, sensores o motores-, esto las hace poco o nada portables, y la flexibilidad que poseen es mínima debido a que el objetivo de la empresa desarrolladora es muchas veces, proporcionar de forma independiente cada uno de los servicios que el cliente requiera.

Como resultado de los esfuerzos de la investigación académica e industrial por eliminar algunos de los problemas ya descritos y otros más específicos –entre ellos los relacionados con seguridad, costos, realización de tareas complejas-, se ha desarrollado lo que hoy se conoce como MRS (Multi-Robot Systems). Los sistemas multi-robot, consisten en plataformas completas donde se pueden implementar varios robots para desarrollar una tarea conjunta, encerrando todos los desafíos que esto conlleva – sensores, actuadores, comunicación, y un largo

etcétera-(Jones et al 2004). Los MRS son una forma particular de los sistemas multi-agente (MAS) centrados en reactividad, colectividad y cooperación para cumplir con tareas de forma que los sistemas sean más robustos, modulares y confiables.

Existen múltiples plataformas de este tipo con dedicación a cierta clase de tarea o robot, por citar algunas: esta ALLIANCE que es un framework para agentes heterogéneos MRS (Parker 1998); SAMON para trabajos con AUV (autonomus underwater vehicles); sistemas colaborativos para fútbol de robots; USARSim para el estudio de tareas colaborativas en USAR (urban search and rescue); EL framework MARTe centrado en el desarrollo de aplicaciones colaborativas en tiempo real (Farinelli et al 2004); MASON presentada como una biblioteca para simulación de multiagentes en java[1]; SMART (García et al 2013), para trabajo con hardware y desarrollo basado en multi-agentes; ROS (Robotic operating system), en el cual se hace énfasis en las siguientes páginas y muchos más.

ROS se convirtió en uno de los sistemas pionero en MRS. Podría decirse que con el objetivo de hacer de la aplicación de la robótica un conjunto de herramientas colaborativas intenta hacerse universal, todo a partir de su flexibilidad. El sistema incluye variadas plataformas, lenguajes, compiladores, y paquetes de aplicaciones desarrolladas por múltiples investigadores, empresas y fanáticos de la robótica para que otros las puedan usar y lo mejor, modificar y crear nuevas y mejoradas opciones. ROS se convierte entonces en un sistema realimentado continuamente, que de acuerdo a los datos crece de manera exponencial en prestaciones y popularidad –en 5 años ha pasado a tener más de 3000 paquetes de aplicaciones y a ser usado en casi 60 plataformas robóticas industriales y de investigación[2]-.

## 2. ROS ROBOT OPERATING SYSTEM.

Aunque su nombre es la sigla para sistema operativo, en realidad ROS es un meta-sistema operativo, ya que aunque ofrece las funciones de un sistema operativo debe ser instalado sobre la base de otro sistema operativo -basado en UNIX-. Las funcionalidades que ofrece van desde la abstracción de hardware de bajo nivel para control de dispositivos, hasta el paso de mensajes entre procesos (Quigley et al 2009).

El objetivo de este sistema operativo es simplificar la tarea de crear cualquier robot imaginable, entregar facilidades para darles funcionalidades y hacerlos robustos y maleables mediante una serie de bibliotecas, drivers y herramientas que como parte de un MRS, le apuestan a la modularidad para ayudar a los desarrolladores. En el sistema se pueden desarrollar paquetes completos por expertos en diferentes materias que pueden ser conjugados en la construcción de un proyecto de gran envergadura.

En ROS cada módulo posee autonomía, de forma similar a lo que sería la comunicación de varios computadores en una red LAN, o en internet en una topología punto a punto, cada módulo interactúa con los otros; esto quiere decir, que todo se hace mediante mensajes enviados de acuerdo a un protocolo, en este caso el XML-RCP, que es un sistema basado en la publicación y registro (Arumugam et al 2010). El uso de este protocolo con amplio soporte facilita que ROS sea una plataforma multilenguaje donde se puede programar primeramente en C++, Python y actualmente en Octave, LISP y hasta java. Pero ROS no se queda allí, sino que hace uso del protocolo TCP/IP para generar un esquema cliente servidor, cuyo principal servidor es el núcleo de ROS, cada nodo tiene una dirección y existe la posibilidad de ofrecer y solicitar servicios (Quigley et al 2009).

Por último pero no menos importante, cabe destacar que el proyecto ROS esta presentado bajo licencia BSD, que permite su uso para desarrollos tanto comerciales como no comerciales (Cousins 2012), reduciendo la inversión y fomentando el uso tanto por desarrolladores independientes como por empresas, dándole un enfoque científico pero aplicado.

Por la estructura, ROS permite una implementación distribuida de elementos tan sencillos o complejos como el desarrollador desee crear; acopla componentes externos -OpenCV, Eigen, Gazebo, Player, etc- (Gerkey et al 2003) en una red completamente funcional, y reduce así, en principio, los problemas de interacción de herramientas a los desarrolladores, permitiéndoles concentrarse en la verdadera innovación.

El núcleo de ROS facilita el manejo de excepciones facilitando la depuración del código; simplifica el manejo del tráfico de mensajes; permite flexibilidad para asignar relevancia a los mensajes; crea sincronización entre elementos mediante sistemas de escucha y espera con actualizaciones constantes (Nguyen et al 2013). En pocas palabras se puede describir como la interacción de múltiples hilos de comportamiento dinámico.

## 3. COMPONENTES DE ROS

[1] Más información en la página de la plataforma http://cs.gmu.edu/~eclab/projects/mason/
[2] Datos en http://.ros.org/

De acuerdo a diversos trabajos los principales componentes de ROS son los siguientes (Rockel 2011), (Tosco et al 2012), (Hevia-Koch 2012), (Quigley et al 2009):

*3.1 Nodo*

Es el elemento estructural de ROS, de forma un poco superficial podría decirse que un nodo equivale a un proceso. Posee hasta 5 elementos básicos que corresponden a definiciones internas, estas son: publicaciones, suscripciones, prestación y petición de servicios, y comportamiento principal (fig. 1.).


*Fig. 1. Estructura de un nodo*

*3.2 Topic*

Son etiquetas usadas como mecanismo para identificar el contenido de un mensaje y generar interacciones con los nodos. Si un nodo publica envía un mensaje a un topic; si un nodo se subscribe recibe un llamado cuando un mensaje de este topic es publicado.
Para que exista un topic debe existir por lo menos un publicador, y para que este tenga razón de ser, debe tener al menos un suscriptor.

*3.3 Mensaje*

Paquete de información emitido por un nodo, escuchado por uno, varios o todos (broadcast), y que viaja en el canal considerado como topic. En el canal pueden viajar varios tipos de mensajes que deben tener el mismo tipo de campos. Los campos son llenados por el publicador y leídos por los suscriptores (fig. 2).

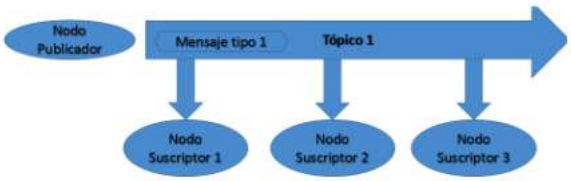



*Fig. 2. Esquema de Mensajes*

*3.4. Servicio*

Entregado bajo una arquitectura cliente-servidor; utiliza un mensaje de solicitud y otro de respuesta. Un único nodo es el prestador del servicio y todos los demás pueden solicitarlo; un nodo prestador de servicios permanece en escucha mientras espera una solicitud.
El tipo de servicio indica la cantidad y tipo de datos que necesita el servicio como parámetros de entrada y la cantidad y tipo de parámetros que envía el servicio como respuesta (fig. 3).


*Fig. 3. Esquema de servicios*

*3.5. Paquete*

Similar a lo que se llamaría un proyecto en un lenguaje de programación básico. Un paquete contiene todo el código fuente de los nodos, las librerías usadas, las cabeceras y cualquier otro recurso necesario para el funcionamiento de la plataforma, la información sobre el tipo de lenguajes usados, servicios prestados, y hasta la forma de compilación (fig. 4).

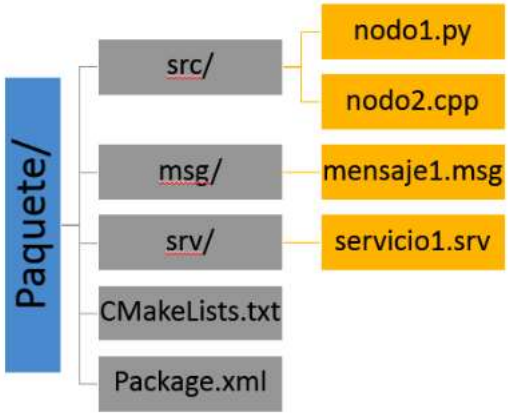



*Fig. 4. Estructura de un paquete*

De acuerdo con la ideología que rige a ROS, se define también una entidad que agrupa los paquetes conocida como pila, y por ultimo otra para la agrupación de pilas conocida como repositorio; los repositorios están disponibles online y se generan continuamente en todo el mundo.

## 4. FUNCIONAMIENTO DE ROS

Luego de creado un paquete y de que se han definido varios nodos dentro del mismo -nodos

que pueden ir desde la simple transformación de una variable hasta el procesamiento de video-, se inician 2 grupos de actividades diferentes:

Un grupo corresponde a la suscripción-publicación. En este caso por ejemplo, un nodo está encargado de publicar la hora cada tiempo determinado en el canal tiempo, entonces, una vez se produce la hora todos los nodos suscritos para recibirla reciben una notificación de que el topic está habilitado e inmediatamente se colocan en escucha para recibir la información, luego retornan a sus labores.

El otro grupo es el de la prestación-petición de servicio. Un servicio puede ser sencillamente la suma de dos números enteros, en este caso, el nodo que emite la petición debe entregar los dos parámetros para que el mensaje sea enviado; una vez que el mensaje llega al nodo prestador o ha recorrido todos los nodos prestadores de servicios se genera un mensaje de respuesta. El mensaje de respuesta puede ser el retorno del servicio, en este caso el resultado de tipo entero; un mensaje de espera, ya que si otros nodos han enviado antes la misma petición se deben atender como una fila –aunque se puede enviar una petición para un servicio del mismo tipo con mayor prioridad, en este caso suma_urgente o algo similar-; en el peor de los casos, un mensaje de que no existe un nodo prestador del servicio. Una vez el nodo solicitante recibe la respuesta retorna al estado anterior al envío del mensaje pero con la información actualizada.

## 5. SIMULACIÓN DE UN SENSOR

Como una introducción al desarrollo en ROS se ha creado un nuevo proyecto con dos nodos, con el objetivo de simular el envío de información desde un sensor hacia un nodo central, que puede ser un robot cualquiera. En la fig. 5. se encuentra el esquema de la red que genera ROS una vez realizado el proyecto mediante el uso de algunos comandos básicos.


*Fig. 5. Diagrama de la red*

La estructura del proyecto creado se muestra en el esquema 1:

```
..\ROB1
|   CMakeLists.txt
|   package.xml
+---msg
|       Muestra.msg
|       Sensor_Mode.msg
+---src
|       sensor.py
|       robot.py
|---srv
|       Sensor_Ready.srv
\
```

*Esquema 1. Estructura del proyecto*

Como se puede observar, en la carpeta principal se encuentran el **MakeFile** donde se invocan los paquetes necesarios de ROS, los mensajes y los servicios usados. En el archivo **package.xml** están las instrucciones de las dependencias que se deben compilar y ejecutar, así mismo, la información básica del proyecto, como lo es nombres, licencias, versión, comentarios, etc.

En la carpeta **msg** se encuentran las configuraciones de los tipos de mensajes que se usan en las diferentes actividades de publicación y suscripción. En este caso el tópico **muestra** solo posee un entero *muestra* y el tópico **Sensor_Mode** posee un booleano *mode* y un flotante *SendRate*.

La carpeta **srv** contiene los tipos de servicios definidos. Para el ejemplo únicamente se tiene el servicio **Sensor_ready**, que retorna un valor *ok* de tipo booleano cuando el servicio se habilita.

En la carpeta **src** están los nodos que hacen uso de los elementos ya creados. En general cada nodo en una primera sección importa las librerías, los mensajes y los servicios que necesita; en una segunda sección, se declaran tanto funciones como variables propias; en una tercera sección se crea la función principal de nodo que contiene definiciones de los servicios, suscripciones y publicaciones, la inicialización del nodo -*rospy.init_node('nodo')*- y el código correspondiente a su proceso principal.

El nodo **sensor** genera el servicio *Sensor_Ready*, está suscrito a *Sensor_Mode* y publica *muestra*. Para simular el comportamiento de un sensor se implementa una función de retardo que mantiene en modo ocupado al nodo, representando el tiempo de adquisición de la señal. Cuando el sensor ha terminado la adquisición, comunica la información al nodo que se encuentre subscrito al servicio (nodo ***robot***), y publica el valor muestreado, que corresponde a un dato aleatorio (fig. 6); el nodo de nuevo se coloca en modo ocupado por otro intervalo de tiempo y se repite la labor indefinidamente.

```
manuel@Gaspar-vm:~$ rosrun rob1 sensor.py
the rosdep view is empty: call 'sudo rosdep init'
[INFO] [WallTime: 1393539849.965170] waiting
[INFO] [WallTime: 1393539850.966542] waiting
[INFO] [WallTime: 1393539851.968032] waiting
[INFO] [WallTime: 1393539852.969617] waiting
[INFO] [WallTime: 1393539853.971160] waiting
[INFO] [WallTime: 1393539854.972731] waiting
[INFO] [WallTime: 1393539855.974335] waiting
[INFO] [WallTime: 1393539856.975916] waiting
[INFO] [WallTime: 1393539857.977486] waiting
[INFO] [WallTime: 1393539858.979008] waiting
[INFO] [WallTime: 1393539860.982314] waiting
[INFO] [WallTime: 1393539861.983196] waiting
[INFO] [WallTime: 1393539862.984716] waiting
[INFO] [WallTime: 1393539863.986108] 4
[INFO] [WallTime: 1393539864.987459] 2
[INFO] [WallTime: 1393539865.988922] 5
```

*Fig. 6. Nodo sensor*

**Robot** está subscrito a *muestra* y publica en *Sensor_mode* -contrario a sensor-, generando así un lazo lógico de comunicación entre los dos nodos, como se establece en la fig. 5.

Robot envía una solicitud de *Sensor_Ready* que le avisa cuando el sensor está disponible para enviar muestras. Como robot está subscrito a la publicación de *muestra* recibe la información cada vez que es generada y la almacena en un buffer. Cuando el buffer está lleno, el robot realiza una operación, en este caso de promedio con los datos y los imprime en la terminal (fig. 7).

```
[INFO] [WallTime: 1393539872.422583] Valor promedio 3.000000
[INFO] [WallTime: 1393539872.422729] Valor promedio 3.000000
[INFO] [WallTime: 1393539872.422887] Valor promedio 3.000000
[INFO] [WallTime: 1393539872.423047] Valor promedio 3.000000
[INFO] [WallTime: 1393539872.513320] Valor promedio 3.000000
[INFO] [WallTime: 1393539872.513496] Valor promedio 3.000000
[INFO] [WallTime: 1393539872.513654] Valor promedio 3.000000
[INFO] [WallTime: 1393539872.513812] Valor promedio 3.000000
[INFO] [WallTime: 1393539872.593619] Valor promedio 3.000000
[INFO] [WallTime: 1393539872.593768] Valor promedio 3.000000
[INFO] [WallTime: 1393539872.593916] Valor promedio 3.000000
[INFO] [WallTime: 1393539872.594067] Valor promedio 3.000000
```

*Fig. 7. Nodo robot*

## 6. IMPLEMENTACIÓN DE UN ROBOT SERIAL

Cuando se diseña y se construye un nuevo robot para ser implementado, se hace necesario utilizar un framework que permita explotar de forma controlada las funcionalidades para las que la planta física fue creada. Existen múltiples ejemplos de plataformas robóticas con hardware y software desarrollado por equipos de trabajo en robótica alrededor del Mundo, los cuales han logrado implementar y/o potenciar en ROS[3]

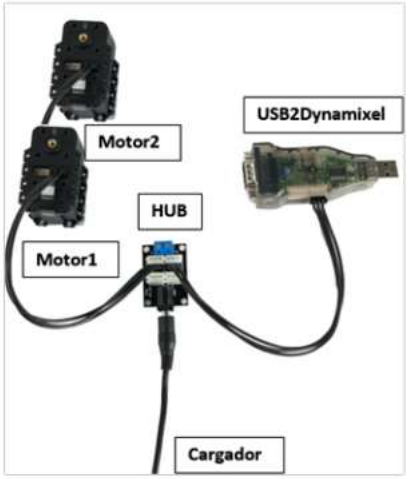
De esta manera, tomando una planta elaborada desde cero, se puede crear toda la red de nodos, servicios y mensajes necesarios para lograr controlar cada una de las acciones del robot desde el nivel de los motores, inclusive, se puede obtener información relevante sobre el estado de la planta, resultados de cálculos y hasta simulaciones en tiempo real, utilizando paquetes adicionales de desarrollos de software como Gazebo y V-REP[4].

### *6.1. Descripción del Hardware*

La plataforma consta de un robot serial de cinco grados de libertad (fig. 8), añadiendo un gripper como efector final; El robot ha sido construido con servomotores Dinamixel de diferentes series que se comunican bajo su propio protocolo[5].

*Fig. 8. Manipulador Serial de 5GDL y efector*

Los Servos se conectan en serie tanto para alimentación como para control. El control se realiza a través de un USB2Dinamixel conectado a un PC y a un Hub donde se conecta el primer motor de la conexión en serie y la fuente de energía (esquema 2).




*Esquema 2. Conexión de hardware*

### *6.2. Descripción de la Aplicación en ROS*

[3] El ejemplo más destacado es RP2 desarrollado por the willow Garage, en http://www.willowgarage.com/pages/software/ros-platform

[4] Más información sobre la plataforma en http://coppeliarobotics.com/

[5] Conceptos básicos y ejemplos en http://www.robotis.com/xe/dynamixel_en

En la fig. 9 se presenta un esquema simplificado como acercamiento a la red implementada en ROS. La estructura contiene nodos generalizados para lograr el funcionamiento adecuado del manipulador:

- El bloque Dynamixel consiste en la comunicación con la planta física. Integran todos los nodos que gestionan y permiten enviar señales que puedan ser interpretadas en principio por el USBDinamixel.

- Nodos específicos para cada motor, que convierten las instrucciones llegadas del resto de la red en instrucciones de movimiento para cada servomotor, en el estándar de los motores, que son enviadas por el bloque de nodos Dynamixel.

- La cinemática del robot se encuentra implementada por completo, por lo que se incluyen nodos de funciones que generan cálculos de cinemática directa, cinemática inversa y Jacobianos.

- Por otro lado para poder obtener todas las variables de interés de acuerdo al nivel de control deseado, se tiene un bloque destinado a los estimadores, donde se incluyen nodos como por ejemplo los filtros Kallman.

- Otra parte gestiona la manera en la que el robot se mueve (Motion control). En esta sección se han contemplado bloques con nodos para la planeación de trayectorias, los interpoladores, e incluso splines.

- Para encargarse la interacción de todas estas partes, se usan nodos dentro de un bloque de control. El bloque comunica directamente a los nodos motores los resultados de todos los procesos internos, interactuando con nodos que hacen parte del bloque controlador, cuya función es administrar los nodos relacionados con la forma de movimiento.

- Por último, existen unos nodos que corresponden a la visualización en una interfaz gráfica de la simulación de la planta en tiempo real, todos incluidos en el bloque interfaz.

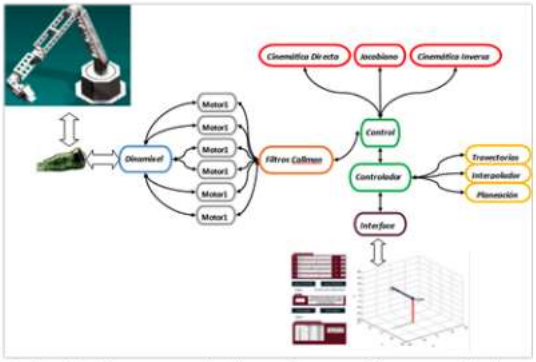



*Fig. 9. Esquema de la red para el manipulador*

En la wiki de ROS[6] existe información sobre algunos paquetes que ya se han desarrollado que han facilitado estas tareas, entre ellos está el paquete tf que proporciona algunas herramientas para realizar cálculos de cinemática en cuanto a marcos de referencia se refiere; otra herramienta es el paquete dinamixel_motor que incluye algunas funciones de manipulación de los motores que posee la planta; entre otros.

## 5. AGRADECIMIENTOS

**A la Universidad Nacional De Colombia, A Través De Las Vicerrectoría de Investigación. PROGRAMA DE FORTALECIMIENTO. PROGRAMA NACIONAL DE SEMILLEROS DE INVESTIGACIÓN, CREACIÓN E INNOVACIÓN DE LA UNIVERSIDAD NACIONAL DE COLOMBIA 20132015. Proyecto 18344 ROBOT A ESCALA TIPO INDUSTRIAL**

## 6. CONCLUSIONES

ROS se visualiza como una herramienta que permite el avance profundo en los proyectos, que fomenta la interacción entre grupos de investigación y que de acuerdo a la situación actual está acoplando la investigación académica e industrial.

Aún existen debilidades en ROS que se pueden minimizar con el tiempo. Cabe mencionar, la falta de un estándar para todo el software que se va implementando; integrar aplicaciones y aprender a usar todas estas herramientas necesita de un periodo de tiempo amplio, soporte para todos los sistemas operativos, entre otras.

Como lo demuestran los casos prácticos, ROS permite realizar desde las aplicaciones más básicas, que pueden ir desde la simple simulación de un proceso, hasta aplicaciones completas que incluyan todas las herramientas que se espera tener cerca, cuando se usa un robot, y lo que es aún mejor múltiples plataformas.

[6] http://wiki.ros.org/

Una implementación como la expuesta con el robot serial facilita que luego se puedan iniciar nuevas investigaciones sobre la misma aplicación, con un mayor nivel de profundidad y en temas tan variados como la colaboración, control, optimización y un largo etcétera.

## REFERENCIAS

Arumugam R., E. V., B. L., X. W., K. K., K. S., (2010), M. K., K. G., *"DAvinCi: A Cloud Computing Framework for Service Robots"*, IEEE International Conference on Robotics and Automation.

Cousins S., (2012). *"Is ROS Good for Robotics";* IEEE Robotics & Automation Magazine.

Farinelli A., I. L., N. D., (2004), *Multi-Robot Systems: A classification focused on coordination,* IEEE Transactions on System Man and Cybernetics, part B.

García C., C. P., S. R., P. L., (2013), *A cooperative multi-agent robotics system: Design and modelling,* artículo en Experts Systems with applications, Science Direct.

Gerkey B., V. R., A. H., (2003), *The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems,* Proceedings of the International Conference on Advanced Robotics (ICAR)., Coimbra, Portugal.

Hevia-Koch P., (2012), *implementación de middleware ROS para robot de servicio, U*niversidad de Chile, Facultad de ciencias físicas y matemáticas, departamento de ingeniería eléctrica, Santiago de Chile, Chile.

Hold-Geoffroy. Y., G. M., G. C., L. M., G. P., (2013), *ros4mat: A Matlab Programming Interface for Remote Operations of ROS-based Robotic Devices in an Educational Context,* international Conference on computer an d Robot Vision.

Jones C., S. D., M. M., G. B., (2004), *Principled Approaches to the Design of Multi-Robot Systems,* Proc. Workshop on Networked Robotics, IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS-04), Sendai, Japón.

Nguyen H., C. M., H. K., K. C., (2013), *"ROS Commander (ROSCo): Behavior Creation for Home Robots"*, ICRA.

Parker, L., (1998), *ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation,* IEEE transactions on robotics and automation, **vol. 14**, no. 2.

Rockel S., (2011), *A Multi-Robot Platform for Mobile Robots with Multi-Agent Technology*, Tesis para el grado de Master del departamento de Informática, Universidad de Hamburgo, Hamburgo, Alemania

Quigley M., G. B., C. K., F. J., F. T., L. J., B. E., W. R., N. A. (2009), *ROS: an open-source Robot Operating System,* Computer Science Department, Stanford University, Stanford, CA.

Tosco S., C. F., B. M., (2012), *"implementación de un esquema de teleoperación utilizando el sistema operativo ros en el contexto de un laboratorio remoto",* Mecánica Computacional **Vol XXXI**, Salta, Argentina.