Ros-Gazebo. una Valiosa Herramienta de Vanguardia para el Desarrollo de la Robótica

Ros-Gazebo. a Valuable Tool Of Avant-Edge for the Development of Robotics

Cristian Camilo Cuevas Castañeda

Escuela de Ciencias Básicas Tecnología e Ingeniería,
Universidad Nacional Abierta y a Distancia, Tunja, Colombia

cccuevasc@unadvirtual.edu.co

Recibido: 10/08/2015 • Aprobado: 12/11/2015

Resumen

El Sistema Operativo Robótico – ROS (de aquí en adelante ROS) representa un significativo avance en la tecnología robótica, ya que constituye un verdadero modelo colaborativo de desarrollo, abierto al público en general y con una gama de posibilidades aún por descubrir. ROS permite contar con estructuras ya diseñadas y programadas que luego se pueden modificar, evitando, de esta manera, comenzar de cero con cada diseño y superando la pérdida de tiempo inherente a la construcción de algoritmos de piezas comunes, como brazos y ruedas, entre otras. Tal plataforma se complementa con las herramientas de Rviz y Gazebo, que brindan simulaciones 3D del modelo robótico diseñado.

Palabras clave: algoritmos, código abierto, entorno de simulación, Linux, programación, ROS, sistema.

Abstract

Robotic Operating System (ROS), represents a significant advance in robotics technology, since it constitutes a true collaborative model of development, open to the general public with a range of possibilities, yet to be discovered. ROS enables structures already designed and scheduled that you can then modify, avoiding in this way, to start from scratch with each design and overcoming the loss of time inherent in the construction of algorithms of common parts, as arms and wheels, among others. Such a platform is complemented with the tools of Rviz and Gazebo, that provide 3D simulations of the robotic model designed.

Key words: *algorithms, linux, open source, programming, ROS system, simulation environment.*

I. Introducción

En esta investigación se expone lo qué es ROS y su importancia, de cara a los avances de la robótica en el mundo. Se trata de un estudio descriptivo que se fundamenta en la revisión del material documental existente de tal sistema en el entorno web construido por su propio creador, así como en la bibliografía que se ha venido generando en años recientes para facilitar su comprensión por parte de la comunidad interesada en acercarse a tan apasionante rama del conocimiento. También se busca brindar al lector, una adecuada ilustración acerca de la forma en la que opera el Sistema, de sus ventajas y de las posibilidades que ofrece frente a los desafíos que en la actualidad plantea la robótica.

II. Desarrollo del Contenido

A. Antecedentes

Desde el año 2000, la Universidad de Stanford, mediante su Laboratorio de Inteligencia Artificial (SAIL, por sus siglas en inglés) [1], llevó a cabo grandes esfuerzos por desarrollar sistemas robóticos accesibles, sin restricciones; esto es, de código abierto y, así, posicionar modelos al alcance de la comunidad propiciando la generación de conocimiento de forma colaborativa en el campo de la robótica. Tales trabajos se hicieron a través de grupos de investigación creados al interior de la Universidad, como el "*Stanford AI Robot* (Stair)" y el "*Personal Robots (PR) Program*" [2]. Estos adelantos fueron retomados e impulsados finalmente por Willow Garage que es básicamente un laboratorio de investigación robótica ubicado en Menlo Park, California, en los Estados Unidos.

Willow Garage es una verdadera incubadora de empresas en este tema y, de forma visionaria, puso su atención en el camino ya adelantado por los programas académicos de la Universidad de Stanford e impulsó la aparición de ROS, en el año 2007. La filosofía de Willow Garage gira en torno a contar con un excelente y motivado capital humano. Esto podría resumirse en las palabras de su primer CEO, Steve Cousins, en una entrevista :

"Mi trabajo consistía en llenar el edificio con gente interesante haciendo cosas interesantes alrededor de la tecnología autónoma" [3].

Así, se trataba de conseguir personas creativas y comprometidas, que amaran lo que hacían, con el objetivo de desarrollar investigación robótica.

En el año 2013, la administración de ROS fue transferida a la *Open Source Robotics Foundation* (OSRF), (Fundación para la programación robótica de Código Abierto), ubicada en la ciudad de San Francisco, Estados Unidos. La misión principal de esta institución y que resume su razón de ser y naturaleza, es la siguiente:

"... apoyar el desarrollo, la distribución y la adopción de software de código abierto para su uso en la investigación de la robótica, la educación y el desarrollo de productos" [4].

Como se puede observar, se trata de una importante iniciativa que busca universalizar el emprendimiento y la investigación de la programación en el ámbito robótico mundial. Tales esfuerzos de universalización han sido canalizados por la fundación precitada, hacia la construcción y perfeccionamiento de dos importantes herramientas, que constituyen el objeto del presente escrito.

En este orden de ideas, los dos principales proyectos supervisados por la OSRF son ROS y Gazebo, un avanzado simulador robot en tercera dimensión [5], aunque, como se verá más adelante, Gazebo tuvo un origen diferente. De tal suerte, que ROS y Gazebo fueron herramientas construidas de forma independiente por desarrolladores diferentes, que luego vinieron a resultar complementarias y administradas, finalmente, por la OSRF.

B. Definición de Ros y de Gazebo.

1) Ros: en primer lugar y con el propósito de ubicar al lector, vale la pena responder al interrogante obvio relativo a: ¿qué es ROS?

ROS es un sistema operativo de programación robótica que ha sido desarrollado en su totalidad para ser usado con el sistema operativo Ubuntu, con base en Linux y que de forma experimental (con algunas falencias y detalles por perfeccionar) funciona con IOS y Windows.

La descripción oficial del sistema nos la da el propio sitio web de ROS, y para guardar plena fidelidad con el autor, se transcribe en el idioma inglés, la definición del mismo:

"ROS es un sistema meta-operativo de código abierto para tu robot. Proporciona los servicios que esperarías de un sistema operativo, incluyendo abstracción de hardware, control de dispositivos de bajo nivel, la implementación de la funcionalidad de uso común, de paso de mensajes entre procesos, y la gestión de paquetes. También proporciona herramientas y bibliotecas para la obtención, la construcción, la escritura y la ejecución de código en varios equipos" [6].

A partir de lo anterior, es posible desglosar un poco la definición expuesta por el desarrollador, con miras a obtener una mayor comprensión de la misma. El sistema meta - operativo hace referencia a que permite manejar, a través de suscripciones a nodos, diferentes tipos de componentes de hardware [7]. El código abierto indica que se trata de un programa o sistema al que se puede acceder para su descarga de forma gratuita. El control de dispositivos de bajo nivel hace referencia a que existe un lazo directo de programación entre el software y la máquina o robot de que se trate; esto es, hace referencia a la vía de comunicación, mas no a su complejidad. Por su parte, la funcionalidad de uso común, el paso de mensajes entre procesos y la gestión de paquetes tiene que ver con las formas propias de operabilidad del sistema ya que ROS funciona con paquetes de configuración de cada función o grupo de funciones del robot, de acuerdo con las características particulares del mismo.

Profundizando un poco en este tema, se tiene que Ros tiene dos niveles de conceptos. En primer lugar, se encuentra el nivel de archivos de sistema: se trata de grupos de paquetes (*packages)*, cada uno de los cuales contiene nodos, los que a su vez describen un proceso de funcionamiento determinado. En segundo lugar, se tiene el nivel de computación gráfica, que se refiere a los comandos útiles *messages*, que pueden ser *publishers* y *susbribers* para intercomunicar a los *packages* previamente diseñados [2].

Así, ROS está basado en un modelo de intercomunicación de nodos, mediante los llamados *publishers* y *suscribers*, que envían y reciben mensajes de movimiento y estructura robótica [9]. Estos nodos también pueden ser categorizados como "componentes reusables", lo que implica que pueden tomarse de una construcción de programación para un modelo robótico y usarse en otras modelaciones distintas [3].

2) Gazebo: el sitio web de la OSRF define lo que debe entenderse por Gazebo, en los siguientes términos [5]:

"La simulación robótica es esencial en la caja de herramientas de cada experto en robótica. Un simulador bien diseñado permite probar rápidamente algoritmos, robots de diseño, y realizar pruebas de regresión utilizando escenarios realistas.

Gazebo *es un simulador 3D multi-robot con dinámica. Ofrece la posibilidad de simular con precisión y eficiencia, diversidad de robots, objetos y sensores en ambientes complejos interiores y exteriores. Gazebo genera, tanto la realimentación realista de sensores, como las interacciones entre los objetos físicamente plausibles, incluida una simulación precisa de la física de cuerpo rígido.*

A tu alcance, es un robusto motor de física, gráficos de alta calidad, y las interfaces gráficas programáticas y convenientes. Lo mejor de todo, Gazebo es gratis y es el soporte de una comunidad vibrante".

A partir de esta definición, se entiende que Gazebo es una herramienta gratuita de simulación, que resulta de vital importancia para la prueba de los algoritmos que van siendo elaborados y configurados y, así, brinda la posibilidad de visibilizar los aciertos y yerros en los que se pueda incurrir en el proceso de desarrollo del robot de que se trate. De esta forma, es posible indicar que la herramienta ROS-Gazebo constituye una completa opción de programación robótica, de código abierto y con una importante gama de posibilidades para emprender desarrollos en el campo de la robótica.

La arquitectura de Gazebo está basada en el motor de simulación ODE (*Open Dynamics Engine*), creado por Russell Smith [11], aunque su desarrollo final se atribuye a Andrew Howard y Nate Koenig, por sus investigaciones en la Universidad del Sur de California [12].

C. Carácter colaborativo y universalista de Ros-Gazebo.

La programación en robótica ha llevado aparejada históricamente una serie de dificultades en virtud de la variedad de formas en que la problemática de su confección puede ser abordada. Así, desarrolladores alrededor del mundo han emprendido esfuerzos aislados por dar mayores horizontes a esta interesante rama del conocimiento, dando pasos carentes de conexión entre sí y apelando a metodologías diversas, generando a la larga una suerte de erosión en el avance de esta ciencia.

Esta diversidad de metodologías ha llevado a que los desarrolladores inviertan grandes cantidades de tiempo y de esfuerzo en la formulación de plataformas y sistemas de base que permitan la operabilidad de los llamados "algoritmos robóticos". Al respecto, [13] ha indicado lo siguiente:

"La interfaz de paso de mensajes de ROS se está convirtiendo en un estándar de facto para la interoperabilidad de software del robot, lo que significa que las interfaces ROS tanto para el hardware más reciente y las implementaciones de algoritmos de corte de borde están disponibles. Por ejemplo, el sitio web enumera cientos de paquetes ROS a disposición del público. Este tipo de interfaz uniforme reduce en gran medida la necesidad de escribir código "pegamento" para conectar las partes existentes.

Por supuesto, ROS no es la única plataforma que ofrece estas capacidades. Lo que es único acerca de ROS, al menos a juicio del autor, es el nivel de apoyo generalizado para este a través de la comunidad de la robótica. Esta "masa crítica" de apoyo hace que sea razonable prever que ROS seguirá evolucionando, ampliado y mejorado en el futuro".

Así las cosas, ROS ha aparecido como una propuesta de solución a esta diversidad regresiva, sirviendo como un entorno tipo sistema operativo, que funciona como una plataforma de unificación del avance mundial en la programación robótica. En este orden de ideas, de acuerdo con importantes académicos que se han dedicado a esta materia, la principal fortaleza de ROS, es su aceptación generalizada y el encauce de esfuerzos en el mundo por el aporte a su avance, desde laboratorios y universidades, hasta el segmento de la comunidad misma interesada en la contribución a la robótica.

Debe indicarse que cuando el autor hace referencia a que el sistema contribuye a evitar la pérdida de tiempo en la confección de paquetes de "código pegamento", está indicando, por ejemplo, que para la instalación de un sensor, ya se cuenta con modelos de código, que pueden ser utilizados de forma directa y que pueden ser objeto de variación, de acuerdo con las necesidades o intereses del diseñador o desarrollador.

Así, se puede señalar que ha sido creada una gama de paquetes ROS para evitar código pegamento y con ello agilizar la construcción y diseño de la estructura robótica deseada. A manera de ejemplo, es posible encontrar paquetes para un brazo o para la instalación de una pinza (*gripper*) y un torso, entre otros:

Con la introducción de ROS (*Robot Operating System*) gracias a Willow Garage, los investigadores e implementadores de la robótica tienen la oportunidad

de acceder, ahora más que nunca, al trabajo de otros investigadores, ayudados por un conocido repositorio virtual llamado *GitHub* [14]. *GitHub* es un sitio web usado mundialmente por desarrolladores de diversas tecnologías de código abierto, para compartir y desarrollar colaborativamente sus avances investigativos: "*Github is how people build software*" [15].

D. Modo de funcionamiento de Ros y Gazebo.

Resulta conveniente ilustrar acerca del funcionamiento de ROS, con base en un ejemplo que permita tener un acercamiento a la forma en que se estructuran sus paquetes de funcionamiento. Como se dijo desde un principio, ROS ha sido diseñado por sus creadores y completado para ejecutarse de forma apropiada con el sistema operativo Ubuntu, basado en Linux, siendo apenas experimentales los desarrollos que han tenido lugar para los entornos IOS y Windows [16].

Ya en el entorno Ubuntu, simplemente debe instalarse ROS, para lo cual, es posible seguir las indicaciones existentes en la página de "ROS *installation*" [17]. Todo el proceso de instalación de Ros y su manejo, se lleva a cabo desde la denominada "Terminal de Ubuntu".

Fig. 1 Imagen de terminal en entorno Ubuntu.

Con fines de contextualización, existen las siguientes versiones de ROS, con sus fechas respectivas de aparición [18]:

Ros Box Turtle. Marzo 2, 2010.
Ros C Turtle. Agosto 2, 2010.
Ros Diamondback. Marzo 2, 2011
Ros Electric Emys. Agosto 30, 2011
Ros Fuerte Turtle. April 23, 2012
Ros Groovy Galapagos. Diciembre 31, 2012
Ros Hydro Medusa. Septiembre 4, 2013
Ros Indigo. Julio 22 de 2014.
Ros Jade Turtle. Mayo 23 de 2015.
Ros Kinetic Kame. Mayo 23 de 2016.

Para el caso que se presentará, se utilizará ROS Indigo, trabajando con una versión Ubuntu 14.04. Pueden existir otras combinaciones de sistema operativo y versión de ROS, pero ha de probarse y asegurarse su operabilidad.

En primer lugar, se crea el llamado *Catkin Work Space*, que se constituye en el espacio de trabajo para la configuración del robot, objeto de este trabajo. Los códigos de programación para tal fin aparecen en la página de ROS, cuyo pantallazo se visibiliza de la siguiente forma:

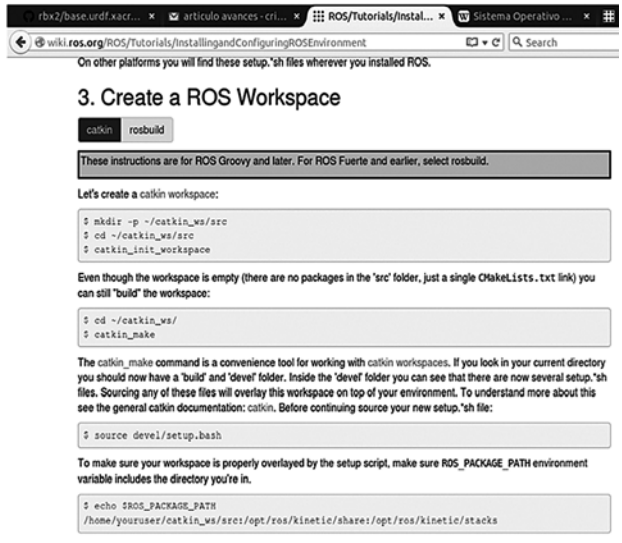

Fig. 2 Tutorial de ROS – Catkin Work Space

Una vez creado el espacio de trabajo, aparecen nuevos directorios, que resultan fundamentales para el trabajo con ROS:

```
workspace_folder/        -- WORKSPACE
  src/                   -- SOURCE SPACE
    CMakeLists.txt       -- 'Toplevel' CMake file, provided by catkin
    package_1/
      CMakeLists.txt     -- CMakeLists.txt file for package_1
      package.xml        -- Package manifest for package_1
    ...
    package_n/
      CMakeLists.txt     -- CMakeLists.txt file for package_n
      package.xml        -- Package manifest for package_n
```

Fig. 3 Tutorial de ROS. Directorios Work Space

Como se puede observar, se crea el directorio *source* (src), que es el que va a contener la gama de archivos que se necesitan para la confección del robot; entre ellos, en *CmakeLists.tx*t y el *package. xml*. Estos archivos son el marco de trabajo para el archivo que finalmente contiene la estructura del robot, esto es, el fundamental UDRF file.

E. UDRF File.

El UDRF *File* es el archivo que contiene el algoritmo de la estructura robótica dentro del entorno ROS. Desde él, se aplican las variaciones a las dimensiones que se quieran y se fijan los componentes pretendidos para el modelo creado.

Para los fines propuestos, se va a tomar como referencia explicativa el ejemplo tomado del libro *ROS by example* 2, simplemente con el ánimo de visibilizar los elementos de un archivo de este tipo y la forma como se determina, desde él, la estructura del robot.

Dado que resulta del todo pertinente, se procede a mostrar el archivo UDRF, que se pasa a explicar:

Bloque 1

```
<?xml version="1.0"?>
 <robot name="base" xmlns:xacro="http://ros.org/
 wiki/xacro">
 <!-- Define a number of dimensions
 using properties →
```

Bloque 2

```
<property name="base_size_x" value="0.30" />
<property name="base_size_y" value="0.30" />
<property name="base_size_z" value="0.12" />
<property name="wheel_length" value="0.02032" />
<property name="wheel_radius" value="0.06191" />
<property name="wheel_offset_x" value="0.09" />
<property name="wheel_offset_y" value="0.17" />
<property name="wheel_offset_z" value="-0.038" />
<property name="PI" value="3.1415" />
<!-- define a wheel →
```

Bloque 3.

```
<macro name="wheel" params="suffix parent
reflect color">
<joint name="${parent}_${suffix}_wheel_joint"
type="continuous">
<axis xyz="0 0 1" />
<limit effort="100" velocity="100"/>
<safety_controller k_velocity="10" />
<origin xyz="${wheel_offset_x} ${reflect*wheel_
offset_y} ${wheel_offset_z}" rpy="${reflect*PI/2}
0 0" />
<parent link="${parent}_link"/>
<child link="${parent}_${suffix}_wheel_link"/>
</joint>
<link name="${parent}_${suffix}_wheel_link">
<visual>
<origin xyz="0 0 0" rpy="0 0 0" />
<geometry>

<cylinder radius="${wheel_radius}"
length="${wheel_length}"/>
</geometry>
<material name="${color}" />
<visual>
</link>
</macro>
```

Bloque 4

```
<!-- The base xacro macro →
<macro name="base" params="name color">
<link name="${name}_link">
<visual>
<origin xyz="0 0 0" rpy="0 0 0" />
<geometry>
<box size="${base_size_x} ${base_size_y}
${base_size_z}" />
</geometry>
<material name="${color}" />
</visual>
```

```
<collision>
<origin xyz="0 0 0" rpy="0 0 0" />
<geometry>
<box size="${base_size_x} ${wheel_offset_y*2 +
wheel_length} ${base_size_z}" />
</geometry>
</collision>
</link>
</macro>
```

Bloque 5

```
<link name="base_footprint">
<visual>
<origin xyz="0 0 0" rpy="0 0 0" />
<geometry>
<box size="0.05 0.05 0.001" />
</geometry>
<material name="TransparentGreen" />
</visual>
</link>
<joint name="base_joint" type="fixed">
<origin xyz="0 0 ${base_size_z/2
- wheel_offset_z}" rpy="0 0 0" />
<parent link="base_footprint"/>
<child link="base_link" />
</joint>
```

Bloque 6

```
<!-- Add the drive wheels →
<wheel parent="base" suffix="l" reflect="1"
color="Orange"/>
<wheel parent="base" suffix="r" reflect="-1"
color="Orange"/>
</robot>
```

Para facilitar la comprensión del archivo UDRF anteriormente mostrado, su estructura ha sido dividida en bloques, con miras a ilustrar de la mejor forma sus componentes y lo que está siendo diseñado y construido en cada uno de los mismos.

Se trata de un algoritmo propio del formato UDRF de un modelo bastante simple, pero que resulta bastante útil para familiarizarse y visibilizar su sintaxis, así como la forma en que se moldean los elementos que se busca, pertenezcan a una estructura robótica determinada.

Antes de continuar, debe señalarse que la herramienta ROS tiene incluida una herramienta de visualización 3D llamada Rviz *3D visualization tool for ROS* [19], que permite observar la forma que va adquiriendo el robot y la manera como él mismo va siendo moldeado de cara a los datos y valores que se van introduciendo en la secuencia.

Desde ya, convendría aclarar que la herramienta Rviz tiene funciones estrictas de visualización y, que a diferencia de la herramienta Gazebo, no prevé esquemas de simulación de fricción, gravedad y otros, ni tampoco es posible la construcción de *worlds* o escenarios de interacción, elementos que sí han sido previstos y adaptados para tener una completa simulación en Gazebo.

Retomando el análisis del ejemplo del archivo UDRF, se puede indicar que el mismo permite visibilizar una estructura en RVIZ, como se muestra en la Fig. 4.

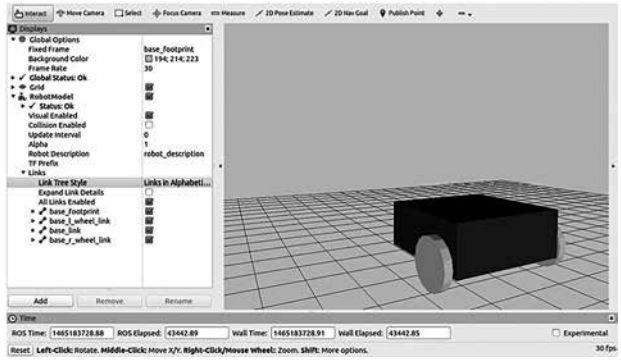

Fig. 4 Estructura visibilizada en RVIZ

Como se venía indicando, se trata de una estructura cúbica bastante simple, con dos ruedas. La misma se visualiza en RVIZ, mediante la apertura de una ventana de terminal distinta y ejecutando el denominado *launch file*, o archivo de lanzamiento. A modo simplemente indicativo e ilustrativo, se puede decir que la estructura de algoritmo *launch file* es la siguiente [20]:

```
<launch>
<rosparam command="load" file="$(find pkg_name)/path/file_name.yaml" />
<node pkg="pkg_name" type="node_type" name="node_name" />
</launch>
```

Para comenzar, se encuentra el primer bloque, que contiene las primeras líneas de todo archivo UDRF, con la única variación posible relativa al nombre que se quiera dar al robot.

En el bloque 2, se encuentran las propiedades de la estructura. Aquí se asignan los valores a las variables que serán usadas en el resto del archivo.

Para comprender este segmento, así como el de otros archivos UDRF que se vayan a analizar, se deben tener en cuenta las siguientes precisiones [5]:

- Las medidas lineares se fijan en metros.
- Los valores angulares se expresan en radianes.
- Los ejes van alineados de la siguiente forma: El eje x apunta en dirección al frente robot; el eje y apunta a la izquierda y el eje z hacia arriba.
- Una articulación rotativa (*roll*) gira en torno al eje x.
- Una articulación prismática (*pitch*) gira en torno al eje y.
- Una articulación de base (*yaw*) gira en torno al eje z.

Así, por ejemplo, las primeras 3 líneas de propiedades indican lo siguiente:

```
<property name="base_size_x" value="0.30" />
<property name="base_size_y" value="0.30" />
<property name="base_size_z" value="0.12" />
```

Corresponden a la estructura cúbica e indican que la medida en el eje (x) es de 30 cm; la medida en el eje (y) también es de 30 cm y en el eje z, de 12 cm.

Seguidamente, el largo de la rueda sería de algo más de 2 cm y su radio de un poco más de 6 cm. Finalmente, las líneas *offset* refieren a la distancia en que las ruedas están montadas, partiendo de la línea de centro del robot:

```
<property name="wheel_length" value="0.02032" />
<property name="wheel_radius" value="0.06191" />
<property name="wheel_offset_x" value="0.09" />
<property name="wheel_offset_y" value="0.17" />
<property name="wheel_offset_z" value="-0.038" />
```

El bloque morado es el correspondiente al macro de la rueda. El llamado componente **xacro macro** refiere a cada uno de los componentes del robot y se asimila a un archivo de c++, el cual debe existir para cada uno de los componentes (ver sitio web: http://wiki.ros.org/urdf/Tutorials/Understanding-PR2URDF).

Cabe indicar que la plataforma ROS trabaja para la elaboración de los archivos UDRF o bien con estructuras de c++ o Phyton [21]. Cuando el nodo algorítmico está escrito en c++ recibe el nombre de *roscpp*, y cuando lo está en Phyton, se denomina *rospy* [22].

Como se puede observar, el posicionamiento de las ruedas se hace estableciendo entre los signos {}, los valores que se habían fijado de forma previa:

```
<origin xyz="${wheel_offset_x} ${reflect*wheel_offset_y} ${wheel_offset_z}" rpy="${reflect*PI/2} 0 0" />
```

El bloque negro corresponde a la base de los archivos macro, que para el caso de esta investigación, corresponde a la estructura cúbica. En el sub-bloque visual se establecen entre los signos {}, las medidas que previamente se habían fijado para el cubo.

```
<box size="${base_size_x} ${base_size_y} ${base_size_z}" />
```

En el sub-bloque *collision* se hace referencia a si se quiere que la estructura cúbica esté algo distanciada

o no de las otras partes. Para este caso, se ha alejado un poco la rueda para evitar atascamientos. Esta distancia dirigida a evitar atascamientos se ha resaltado y corresponde dentro de la siguiente sintaxis a y*2, :

box size="${base_size_x} ${wheel_offset_y*2 + wheel_length} ${base_size_z}

El bloque azul claro denominado *footprint* busca fijar la distancia del plano base; es decir, del suelo. Así, en el ejemplo que se viene manejando es la distancia de la estructura cúbica respecto del suelo.

Finalmente, en el bloque seis se unen las ruedas a la estructura y se les da el color de elección. El parámetro *suffix* le da a las ruedas una denominación diferenciada:

Esta sería la rueda izquierda:

<wheel parent="base" suffix="l" reflect="1" color="Orange"/>

Esta sería la rueda derecha:

<wheel parent="base" suffix="r" reflect="-1" color="Orange"/>

También se puede visibilizar los ejes existentes en cada una de las piezas de la estructura ver Fig. 5.

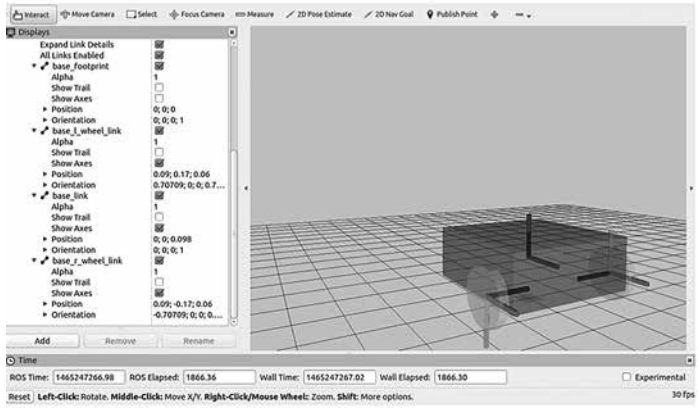

Fig. 5 Ejes x, y, z de la estructura

De igual forma, resulta pertinente indicar que cada eje tiene un color distintivo y específico.

Así, los ejes tienen los siguientes colores: el rojo corresponde al eje x; el verde, al eje y y el azul es el eje z. [23].

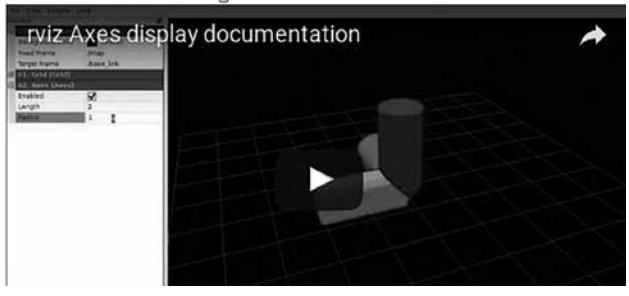

Fig. 6 Colores de los ejes x, y, z

A continuación se visualiza el *offset* de la rueda sobre el eje z:

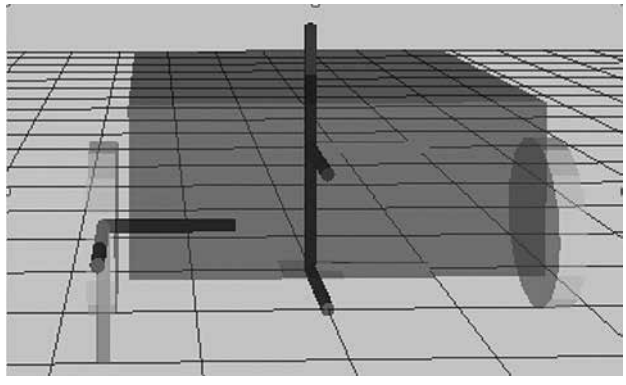

Fig. 7 Offset de la rueda izquierda desde la perspectiva del eje z

A su turno, a continuación, se pasa a visualizar el *offset* sobre los ejes x y y:

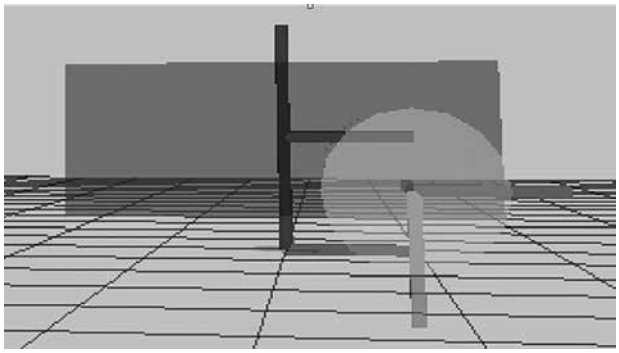

Fig. 8 Offset de desde la perspectiva de los ejes x, y

F. Visualización en Gazebo

Como se ha venido indicando hasta el momento, Gazebo es una simulador 3D, con variadas funcionalidades y que resulta de vital importancia para el aprovechamiento máximo de un sistema como ROS.

Previamente debe indicarse que en un primer momento, Gazebo se creó y desarrolló para trabajar con *Player*, una plataforma - proyecto de desarrollo robótico [24]. Posteriormente, fue adaptado a ROS y en la actualidad se trata de un sistema que bien puede funcionar con desarrollo algorítmico vía ROS o con otros sistemas operativos [25].

Para empezar, se puede señalar que el entorno Gazebo permite de forma directa, crear **mundos** y **objetos** idóneos para insertar el modelo robótico que se esté desarrollando, con el fin de visibilizar la manera en la que el mismo interactúa en un ambiente determinado, pudiendo entonces verificar el funcionamiento de sensores, aptitudes de movilidad e incluso de superación de obstáculos. También resulta posible observar cómo la gravedad afecta la estructura, caídas desde alturas determinadas, fricción, entre otras. Se trata de un sistema con un robusto motor de física y con una alta calidad en la representación de gráficos [26].

Gazebo trae incorporados una serie de *worlds* en los que se puede insertar la estructura robótica y ver cómo se produce su interacción dentro de los mismos. Igualmente, es posible introducir estructuras de construcción como bloques y esferas, entre otras, con miras a visibilizar cómo el robot manipula o traslada objetos de un lugar a otro. El simulador gazebo es atractivo y se usa en todo el mundo en la comunidad robótica porque permite modelar mundos complejos y sensores de visión bastante útiles para visibilizar el funcionamiento del robot [27].

Para ejemplificar, se tomará el denominado *Empty World*, en español, mundo vacío, al que se accede digitando el siguiente comando, desde la terminal de Ubuntu:

```
roslaunch gazebo_ros empty_world.launch
```

En la Fig. 9 es posible visibilizar el *World*, *Gas Station*. Al lado izquierdo se muestra un listado de algunos objetos que vienen incorporados en el programa con miras a facilitar y posibilitar un buen escenario de simulación.

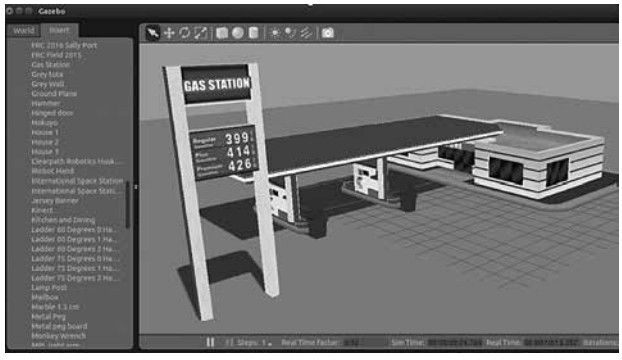

Fig. 9 World in Gazebo: Gas Station.

Igualmente, es posible insertar robots ya construidos y bastante avanzados como el PR2, diseñado y desarrollado por *Willow Garage*, y que en la actualidad sirve como fuente de referencia para la experimentación con Robots en escenarios reales:

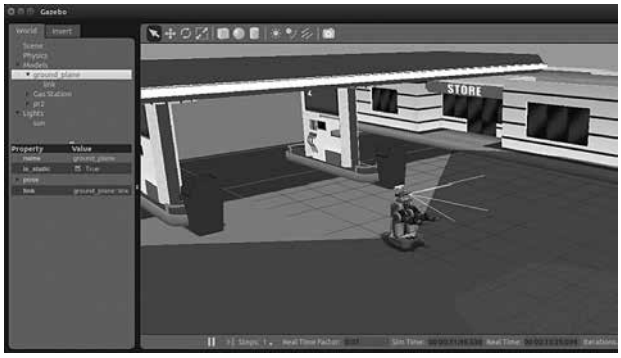

Fig. 10 PR2 en el World Gas Station.

A título informativo, resulta pertinente presentar la descripción del PR2, que se encuentra en el sitio virtual de ROS:

> *El PR2 es una plataforma de manipulación móvil construida por Willow Garage. El sistema de software PR2 está escrito completamente en ROS. Como tal, todas las capacidades PR2 están disponibles a través de interfaces de ROS*.

Es tal la fidelidad de la simulación, que experimentos con robots reales han arrojado la siguiente conclusión, tratándose de ROS y Gazebo:

> *"En experimentos de vida real, esto es equivalente a la observación del comportamiento físico del robot para ver si responde como se espera" [28].*

También es importante indicar que con el binomio ROS-Gazebo es posible realizar códigos de verificación del correcto funcionamiento del robot, mediante alertas de fallas detectadas [29]. Ejemplo de ello, es el código de comprobación de estabilización PID para un robot humanoide, utilizado en el entorno Gazebo [30]. Gazebo también ha sido usado en otros ámbitos de simulación, como la célebre *Robot World Cup*, celebrada cada año [31].

Igualmente, la verificación de los tiempos de movimiento en el campo de la robótica es bastante importante y esta herramienta proporciona elementos de verificación de medidas cronológicas. Ejemplo de ello, son las pruebas efectuadas en los tiempos de colisión del vehículo aéreo no tripulado *Parrot ar drone* [32].

G. Perspectivas actuales de trabajo con ROS.

Para finalizar, y con el ánimo de mostrar algunas perspectivas de la industria y de la dirección que está tomando el trabajo con la plataforma ROS, se puede fijar la mirada en algunos nichos de inversión que tienden a valerse de robots para diferentes actividades y que están demandando su adecuada programación con fines específicos. Debe indicarse también que ROS puede utilizarse en cualquier estructura o proyecto de tipo robótico, sea cual fuere su destinación o funcionalidad:

> *…el potencial de ROS queda ilustrado con la habilidad de trabajar con todo tipo de Robot en un entorno simulado* [33].

1) Drones: recientemente (31 de mayo de 2016), en su cuenta de Twitter, la OSRF compartió un interesante artículo titulado *Why developers should care about the drone industry?* de la revista SDT *Times – Software Development News*, en que se trata el tema relativo al *boom* de la programación robótica en drones.

De la lectura se extracta que lo importante de trabajar con drones, tiene que ver con la información que puede ser obtenida a través de su uso:

> *"Los drones no son interesantes por sí mismos; lo que es interesante acerca de los drones es lo que pueden hacer: los datos. Lo que pueden hacer estarà definido por el Software", dijo Chris Anderson, CEO de 3D Robotics (3DR), un proveedor, fabricante de aviones con tecnología de no tripulados* [34].

Así, los drones no resultan de interés por sí mismos, sino por lo datos que pueden captarse con su manejo y conducción a través de programas especializados de configuración informática:

> *Drone image processing requires specialty configured compute resources* [35].

Dado el aumento desmesurado de estos artefactos en diferentes lugares del mundo, se ha vuelto imprescindible la búsqueda de soluciones a través de la programación, para lograr su operación segura durante actividades de vuelo:

"Los desarrolladores no sólo tendrán la tarea de ser capaces de recolectar, rastrear, analizar, gestionar y dar sentido a los datos del Software, sino también asegurarse de que estos vehículos no tripulados puedan operar con seguridad en el cielo" [34].

Así mismo, los lenguajes de programación que se utilizan actualmente para la programación con drones, los mismos que utiliza ROS para la construcción algorítmica, son: C++ y Phyton.

Los desarrolladores de lenguajes de programación más populares que están utilizando para crear software de drones, a menudo usan el C o C ++ debido a su flexibilidad y bajo nivel de control, de acuerdo con Bowen.

"Sin embargo, si desea ejecutar una aplicación en la parte superior del código de vuelo a través de APIs, Python es un poco arriesgado". Dijo Anderson de 3DR: "Podrías escribir en C o Java, pero Python es el lenguaje de acceso para funciones de nivel superior" [34].

Los campos previstos por la industria para la utilización de drones realmente son muy variados e incluyen:

- La construcción de soluciones para ayudar a los agricultores a medir sus cultivos y controlar la vegetación, dentro de la llamada agricultura de precisión. Esto se logra a través de la reducción de costos de operación y mejorando el rendimiento con la mayor generación de alimentos con menos insumos, frente a la demanda de la creciente población mundial. [36]. Se proyecta un importante crecimiento de esta industria en el mundo: *"The worldwide market for agricultural drones is $494 million anticipated to reach $3.69 billion by 2022"* [37]. Así, el mercado global para drones de agricultura crecerá de una valuación de los 494 millones de dólares a los 3,69 billones en 2022.

- Desarrollo de un sistema que pueda ayudar a los esfuerzos de rescate en situaciones de emergencia. Ejemplo de ello, es el sistema S.W.A.R.M. (*Search With Aerial Rc Multi-otor*), dirigido a la búsqueda de personas desaparecidas [38].

- Implementación de señal de internet, mediante drones, en lugares alejados. Ya está siendo usado por la empresa Facebook, en diferentes lugares de América Latina, África y Asia [39].

- Ayuda en el control de incendios forestales, aportando a su control y predicción mediante drones manipulados desde bases terrestres [40].

2) Avances en el área de investigación espacial: en el aspecto de investigación espacial, *Robonaut* ha sido un proyecto respaldado por la NASA, en colaboración con la *General Motors*, el cual es presentado en su página oficial en los siguientes términos:

Robonaut es un hábil robot humanoide construido y diseñado en el Centro Espacial Johnson, en Houston, Texas [41].

Este robot humanoide se encuentra en la Estación Espacial Internacional y desarrolla tareas de forma permanente en tal entorno. La palabra ***dexterous*** hace referencia a que sus manos y dedos se mueven como los de un ser humano:

Robonaut es llamado un robot dexterous porque sus manos y dedos se mueven como los de una persona. Así que Robonaut puede realizar tareas diseñadas para ser hechas realizadas por manos humanas. Por ejemplo, Robonaut puede utilizar muchas de las mismas herramientas que usa un astronauta [42].

Este Robot fue implementado en su programación a nivel íntegro en la plataforma ROS y todos sus componentes pueden ser hallados en el sitio Web de tal plataforma.

El estudio de las particularidades de programación de este magnífico robot en la plataforma ROS se encuentra en el capítulo del libro *ROS in Space: A Case Study ON Robonaut 2*.

Se trata del abordaje del caso de estudio planteado por la construcción de este Robot, para la identificación de las nuevas capacidades del software, interfaces de usuario, implementación y operación remota, así como la ruta de aseguramiento de seguridad para su operabilidad en el espacio exterior.

Cabe preguntarse acerca de su uso en tal estación espacial: en primer lugar, ayudará a los astronautas a reducir el gasto de tiempo en labores de rutina de limpieza y mantenimiento de dispositivos. En segundo lugar, la experiencia ganada por el robot en su trabajo será esencial en el diseño de sistemas de guiado para operaciones llevadas a cabo en vehículos de reparación externa [43].

Fig. 11 Robonaut.

Este robot también ha sido acondicionado para colaborar en operaciones de reparación de la Estación Espacial Internacional dentro o fuera de la misma.

Robonaut 2 ha recibido un par de piernas que ayudarán a que se mueva alrededor de la estación, y, finalmente, permitir que el robot trabaje en las reparaciones, tanto dentro como fuera de la estación orbital [44].

III. Resultados y Análisis

En este apartado se presenta una tabla comparativa de ROS (Tabla 1) respecto de otros programas o entornos dirigidos al desarrollo robótico:

TABLA I

Comparativo Entre Simuladores

ROS (ROBOTIC OPERATIVE SYSTEM)	ROBOCELL	SIMUROB	VREP
Entorno en el que funciona: Linux	Entorno en el que funciona: Windows	Entorno en el que funciona: Windows	Entorno en el que funciona: Mac y Linux.
Trabaja con el sistema de simulación Gazebo que permite interactuar con variables como la gravedad, la fricción, el viento y otras no disponibles en programas diferentes: (*odometry, localizatiion, perception* [45]). Odometría hace referencia a localización y variación de movimiento de sensores.	El sistema de simulación no cuenta con variables como viento, rozamiento, fricción, etc.	El sistema de simulación no cuenta con variables como viento, rozamiento, fricción, etc.	Tiene su propio sistema de simulación, aunque también puede enlazarse con ROS y así, trabajar con el sistema de simulación Gazebo que permite interactuar con variables como la gravedad, la fricción, el viento y otras no disponibles en programas diferentes.
Permite la interacción entre modelos creados por diferentes autores. Se trata de una plataforma eminentemente colaborativa	No permite interacción continua entre modelos creados por otros autores. Solo con modelos que ya vienen predeterminados en el programa	No permite interacción continua entre modelos creados por otros autores. Solo con modelos que ya vienen predeterminados en el programa	Permite la interacción entre modelos creados por diferentes autores. Se trata de una plataforma eminentemente colaborativa
Uno es el entorno donde se trabaja en la configuración y estructuración del robot y otro distinto, en el que se accede a la simulación con Gazebo.	Dentro de un mismo entorno, permite trabajar en las órdenes al robot, mediante instrucciones previamente determinadas	Dentro de un mismo entorno, permite trabajar en las órdenes al robot, mediante instrucciones previamente determinadas	Dentro de un mismo entorno permite trabajar en las órdenes al robot, mediante instrucciones previamente determinadas
Las instrucciones se definen con base en comandos algorítmicos que siguen una estructura determinada, y que pueden construirse y determinarse, de acuerdo con las necesidades y destinaciones. [46]	Las instrucciones se hacen con base en comandos que ya vienen creados en el entorno del programa. [47]	Las instrucciones se hacen con base en comandos que ya vienen creados en el entorno del programa. [48]	Las instrucciones se definen con base en comandos algorítmicos que siguen una estructura determinada, y que pueden construirse y determinarse, de acuerdo con las necesidades y destinaciones [49]

Vale señalar que ROS ha sido notablemente exitoso en el ámbito investigativo y comercial del mundo de la robótica. Sin perder de vista el carácter reciente de este sistema, para el año 2015, existían cerca de 2.000 paquetes levantados en esta plataforma de software, alrededor de 80 robots disponibles comercialmente basados en ROS y 1.850 artículos investigativos mencionando a ROS en sus contenidos [50].

De esta manera, el resultado investigativo refiere a que se ha descubierto una herramienta de software sumamente importante para al avance de la robótica en el país, cuyo uso debe promoverse, ya que como resulta visible en el grueso de la referenciación bibliográfica consultada, se trata del sistema que, en la actualidad, marca la tendencia del desarrollo robótico en naciones desarrolladas.

Desde la perspectiva del paradigma de Colombia, es dable considerar que estas iniciativas pueden dar una respuesta a los interrogantes referidos a la forma como debe desarrollarse el sector agrícola colombiano, representando la robótica un esperanzador camino a la tecnificación del mismo, tal y como ha venido ocurriendo en naciones principalmente europeas desde hace años. Ejemplo de ello, es el proyecto ICT AGRI QUAD-AV (*Ambient*

Awareness for Autonomous Agricultural Vehicles), financiado por la Agencia Danesa de la Industria de Alimentos [51].

IV. Conclusiones y Trabajos Futuros

ROS se perfila como una opción mundial para el avance de la programación robótica. En este sentido, resulta importante desarrollar competencias en el manejo de tal sistema, yaque si se pretende, en un futuro, presentar propuestas a problemas concretos como el atinente al manejo adecuado de drones con diferentes propósitos, resulta fundamental usar a la perfección este tipo de herramientas.

El sistema ROS se encuentra en constante desarrollo y evolución, por lo que el adecuado estudio del mismo, representa una tarea permanente e inagotable.

Son grandes los desafíos que el mundo actual impone a quienes les apasiona el campo de la programación robótica. Existen valiosas herramientas de código abierto que están al alcance del público, siendo ROS una de ellas. Tales herramientas se encuentran ahí y están a la espera de nutrirse de contenidos que aporten al avance de tan importante ramo del conocimiento, en pro de la contribución al progreso de la humanidad.

Referencias

[1] Stanford University, *Stanford Artificial Intelligence Laboratory (SAIL)*, (2016, Sep 08). [Online]. Available: http://ai.stanford.edu/.

[2] OSRF, *ROS*, (2013). [Online]. Available: http://www.ros.org/history/. [Último acceso: 15 06 2016].

[3] J. D´Onfro, *How a billionaire who wrote Google's original code created a robot revolution*, Bussiner Insider, 2016.

[4] Willowgarage, *Willowgarage*, 2008. [Online]. Available: https://www.willowgarage.com/pages/about-us. [Último acceso: 15 06 2016].

[5] P. Goebel, "ROS by example", Vol. 2. Packages and Programs for Advanced Robot Behaviors, California, 2014.

[6] OSRF, *ROS*, (2013). [Online]. Available: http://wiki.ros.org/ROS/Introduction. [Último acceso: 15 06 2016].

[7] F. Paz, *The Corpora Robotic Company*, (2010). [Online]. Available: http://thecorpora.com/blog/?p=335&lang=es. [Último acceso: 2016 06 15].

[8] A. Goncalves, *ROSInt - Integration of a mobile robot in ROS architecture*, Coimbra: University of Coimbra, 2016, p. 27.

[9] G. Parisi y S. Wermter, *A Neurocognitive Robot Assistant for Robust Event Detection*, Trents in ambient intelligent systems. Studies in computational intelligence, pp. 1-29, 2016.

[10] O. Khatib , Handbook of Robotics, 2 ed., Stanford: Springer, 2016.

[11] P. Gómez del Torno, O. Alvarez fres y S. Marcos Pablos, *Robotic Development* de *Service Robotic within the Digital Home*, New York, Springer, 2011, pp. 50-88.

[12] M. Eaton, Evolutionary Humanoid Robotics, Limerick: Springer, 2015.

[13] J. O´ Kane , A Gentle Introduction to Ros, Columbia: University of South Carolina, 2014.

[14] T. T. Andersen , *Optimizing the Universal Robots ROS driver*, Technical University of Denmark, 2015.

[15] T. Preston-Werner y C. Wanstrath, *GITHUB*, (2010). [Online]. Available: https://github.com/. [Último acceso: 12 09 2016].

[16] S. Puligny, *ROS interface and URDF parser for Webots*, Ecole Polytechnyque Federale de Lausanne, Lausanne, 2014.

[17] OSRF, *ROS*, (2013). [Online]. Available: http://wiki.ros.org/indigo/Installation/Ubuntu. [Último acceso: 15 06 2016].

[18] OSRF, *ROS*, (2013). [Online]. Available: http://wiki.ros.org/Distributions. [Último acceso: 15 06 2016].

[19] OSRF, *Ros Rviz*, (2013). [Online]. Available: http://wiki.ros.org/rviz. [Último acceso: 15 06 2016].

[20] Stack Over Flow, *How to use an launch file in xml document to get the parameters to be used in a cpp file*, (2016). [Online]. Available: http://stackoverflow.com/questions/29493361/how-to-use-an-launch-file-in-xml-document-to-get-the-parameters-to-be-used-in-a. [Último acceso: 08 09 2016].

[21] J. Lentin, Mastering Ros for Robotics Programming, Birmingham: Packt Publishing, 2015, p. 3.

[22] A. Goncalves, *ROSint - Integration of a mobile robot in ROS architecture*, Coimbra: University of Coimbra, 2012, p. 78.

[23] OSRF, *ROS Axes*, (2013). [Online]. Available: http://wiki.ros.org/rviz/DisplayTypes/Axes. [Último acceso: 15 06 2016].

[24] Player Project, (2014). [Online]. Available: http://playerstage.sourceforge.net/. [Último acceso: 09 09 2016].

[25] A. Nuno Dos Santos, *RobotTeamSim – 3D Visualization of Cooperative Mobile Robot Missions in Gazebo Virtual Environment*, Coimbra: University of Coimbra, 2013.

[26] J. J. López Perez, V. Ayala-Ramírez y U. Hernández-Belmonte, «Dynamic Objetc Detection and Representation for Mobile,» de Pattern Recognition. 8th Mexican Conference. MCPR 2016. Guanajuato, México. June 22-25, 2016. Proceedings, 2016.

[27] T. Habra , D. Houman, A. Cardellino, L. Natale, N. Tsagarakis, P. Fisette y R. Ronsse, *ROBOTRAN-YARP Interface. A Frame World for a Real Time Controller Developments Based on Multibody Dynamics Simulations*, Computational Methods in Applied Sciences, pp. 147-164, 2016.

[28] D. Araiza-Illan, D. Western, A. Pipe y K. Eder, «Verification of Robotics. Coverage-Driven Verification - An approach to verify Code for Robots that directly interact with humans,» de Hardware and Software: Verification and Testing. 11th International Haifa Verification Conference, HVC 2015 Haifa Israel. November 17-19, 2015. Proceedings , 2015.

[29] D. araiza-Illan, D. Western, A. G. Pipe y K. Eder, «Systematic and Realistic Testing in Simulation of Control Code for Robots in Collaborative Human Robot - Interactions,» de Towards Autonomous Robotic Systems. 17th Annual Conference, TAROS 2016 Sheffield, UK, June 26 - July 1, 2016. *Proceedings*, Sheffield.

[30] B. Madhu, K. Surya, H. Roshan kumar y K. Cheruvu Shiv, «Stabilization of Posture of Humanoid Using PID Controller in Gazebo Simulator Using Robot Operative System (ROS).,» de CAD/CAM Robotics and Factories of the future. Proceedings of the 28th international conference on CARS & FOF 2016, Kolaghat, 2016.

[31] R. A.C. Bianchi, L. Akin, S. Ramamoorthy y K. Sugiura, *Robocup 2014: Robot World Cup*, New York: Springer, Robocup 2014: Robot World Cup XVIII, p. 228.

[32] W. Amalraj Arokiasami, T. K. Chen, D. Srinivasan y P. Vadakkepat, «Impact of the Lenght of Optical Flow Vectors in Estimating Time To Contact an Obstacle,» de Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, Springer, 2015.

[33] A. Martínez y E. Fernández , *Learning Ros For Robotics Programming*, Birmingham: Packt Publishing, 2013.

[34] C. Mulligan, *Why developers should care about the drone industry*, SD Times Software Developers News, 2016.

[35] M. Woodall, *Drone Data*, (2014, 08). [Online]. Available: http://www.dronedata.com/. [Último acceso: 06 09 2016].

[36] J.P. Muller,. Stratfor, *The Fertile common ground between Technology and Agriculture*, (2016, Ago 08). [Online]. Available: https://www.geneticliteracyproject.org/2016/08/09/big-data-romances-big-ag-future-precision-agriculture-may-drones/#link. [Último acceso: 07 09 2016].

[37] Market Research Reports.biz, *Market Research Reports. biz*, (2016). [Online]. Available: http://www.marketresearchreports.biz/. [Último acceso: 08 09 2016].

[38] [38] Swarm. *Volunteer Search & Rescue Network*, (2016). [Online]. Available: sardrones.org. [Último acceso: 12 09 2016].

[39] S. Leal, *e-Renovarse o Morir. 7 tendencias tecnológicas para convertirse en un lider digital*, Barcelona, 2013, p. 40.

[40] F. Camacho Obregón, *Proyecto Drone Fire,* Cadiz: Universidad de Cadiz, 2015.

[41] NASA National Spacial Agency, *R2 Robonaut 2*, (2013). [Online]. Available: http://robonaut.jsc.nasa.gov/default.asp#panel-1. [Último acceso: 23 06 2016].

[42] NASA, *What is Robonaut*, (2012). [Online]. Available: http://www.nasa.gov/audience/forstudents/5-8/features/nasa-knows/what-is-robonaut-58.html. [Último acceso: 23 06 2016].

[43] J. Badger, D. Gooding, K. Ensley, K. Hambuchen y A. Thackston, «ROS in Space: A Case Study on Robonaut 2,» de *Studies in Computational Intelligence*, vol. 625, 2016, pp. 343-373.

[44] E. Howell, *Space.com*, (2014, Sep 09). [Online]. Available: http://www.space.com/27161-space-station-robonaut-legs.html. [Último acceso: 2016].

[45] M. Aznar , F. Gómez_Bravo, M. Sánchez, J. M. Martín y R. Jiménez, «Ros Methodology to Work with Non-Ros Mobile Robots: experimental uses in Mobile Robotics Teaching,» *ROBOT2013: First Iberian Robotics Conference: Advances in Robotics, Volume 2,* vol. 2, nº 253, pp. 411-426, 2014.

[46] O. S. R. OSRF Foundation, *Robotic Operative System*. ROS, 2015. [Online]. Available: www.ros.org.

[47] Intelitek, Robocell Manual, 2005.

[48] M. Beltrán Blanco, Simurob, Manual de Usuario, 2012.

[49] Coppelia Robotics, *V-REP virtual robot experimentation platform*, (2016). [Online]. Available: http://www.coppeliarobotics.com.

[50] M. Quigley, B. Gerkey y W. Smart, Programming Robots with ROS. *A practical introduction to the Robot Operative System*, Sebastopol , California: O´Really Media, 2015.

[51] Robotikka, *Actualidad Gadged*, (2011, Nov 21). [Online]. Available: http://www.actualidadgadget.com/europa-apuesta-por-la-robotica-agricola/. [Último acceso: 13 09 2016].