Memorias del Congreso Nacional de Control Automático
ISSN: 2594-2492

Unificación de sistemas para el control de un robot bípedo con ROS

Adrián Ricárdez Ortigosa * **Edmundo Rocha Cózatl** **

* *(e-mail: adricort.unam@gmail.com)*
** *Departamento de Ingeniería Mecatrónica, Facultad de Ingeniería, UNAM, Mexico City, Mexico (e-mail: e.rocha.cozatl@comunidad.unam.mx)*

Resumen
En esta contribución se propuso una metodología para la implementación un sistema con arquitectura *Peer-to-Peer*, tomando como caso de estudio un robot bípedo de 12 GDL (grados de libertad). Bajo esta arquitectura y utilizando la herramienta de software ROS (Robot Operating System) se modulariza y unifican los procesos para el control de postura y marcha PID autosintonizable de un trabajo previo, el cual utilizaba una arquitectura convencional *BlackBoard*. Los resultados experimentales indican que se lograron ventajas en la implantación física del sistema y abre posibilidades para la mejora de su funcionamiento.

Keywords: Robot bípedo, software ROS, implantación de controladores.

1. INTRODUCCIÓN

La robótica bípeda es un área de la ingeniería que ha tenido un amplio desarrollo en los últimos años como puede revisarse en la literatura (Siciliano (2008) y en la información que proporcionan empresas privadas (Boston Dynamics (2018)).

En particular, en nuestro grupo de trabajo, se ha puesto enfoque en un robot bípedo modelo Scout de Lynxmotion, desarrollado y mejorado por estudiantes de licenciatura y maestría de la Facultad de Ingeniería en las instalaciones del Centro de Ingeniería Avanzada de la UNAM. Uno de los estudios más recientes de grupo han sido en la estabilización de la postura y caminata bípeda (Sánchez (2017)), la cual es escencial, ya que con ella se busca que el robot no caiga durante la caminata o por efecto de alguna perturbación como el cambio en la inclinación de la superficie por donde se desplaza.

El robot con el que se ha trabajado tiene una altura de 23 (cm) y pesa 1.1 (kg); consta de 12 articulaciones, 6 en cada pierna, cada una de ellas accionada mediante servomotores (Sánchez (2017)). Este dispositivo se ha desarrollado por medio de diferentes tipos de sensores y con un arreglo de tarjetas de desarrollo (Arduino, Tiva, Teensy), con las cuales se implanta una arquitectura de programación y procesamiento centralizada llamada por algunos como *Blackboard*. Sin embargo, en su funcionamiento se registraron diversos problemas en el monitoreo de datos, la desconexión de módulos y la desincronización entre los programas que integran el sistema de control.

Por ello, se propone una alternativa para dar solución a este tipo de problemas por medio de un cambio de paradigma en la arquitectura de programación y procesamiento, siendo la llamada arquitectura *Peer-to-Peer* una opción viable, debido a su modularidad y procesamiento eficiente. Un software que funciona bajo esta arquitectura es ROS (Robot Operating System).

Figura 1. Robot bípedo Scout de Lynxmotion™y ROS Kinetic Kame (ROS Wiki , 2018)

El objetivo principal de ROS es simplificar el trabajo de manipular el comportamiento de un robot complejo a través de la diversidad de módulos y plataformas estandarizadas. Además, tiene un gran campo de aplicación dentro de la ingeniería y la industria. ROS, a pesar de llevar poco más de 8 años, es una herramienta solución para diversos proyectos en prestigiados centros de investigación y universidades internacionales, como los robots BigDog (cuadrúpedo) y Atlas (humanoide) de Boston Dynamics (Boston Dynamics , 2018), y Biped Robin, bípedo del Institute for Robotics at Johannes Kepler University Linz (Biped Robin , 2015). Sin embargo, debido a la protección por derechos de autor de estos desarrollos, no se tiene documentación sobre la forma de como se implanta el control con ROS.

Por lo tanto, la aportación de este trabajo es la propuesta de una metodología para la implantación de este tipo de arquitectura en sistemas de control, la cual se basó en la información recopilada de varios proyectos, artículos, libros y documentación disponible de ROS. En particular, se plantea la programación del mismo controlador que en (Sánchez (2017)) bajo este nuevo paradigma. Esta metodología puede ser adaptada a otros proyectos y con esto reducir el tiempo de investigación previa, ya que no

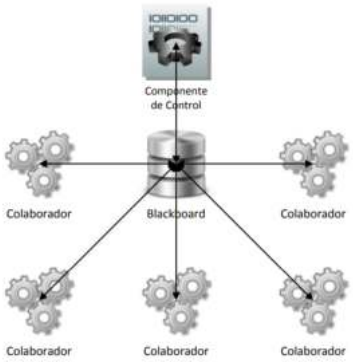

Figura 2. Arquitectura centralizada Blackboard (Matamoros , 2013)

todas las referencias (aparte de algunos libros, artículos o la ROS Wiki) suelen contar con explicaciones claras y concisas de los pasos a seguir, tal y como cita Jason O'Kane en (O'Kane , 2017):

"*Después de todo, aprender a usar un nuevo framework, particularmente uno tan complejo y diverso como ROS, puede tomar bastante tiempo y energía mental, por lo que uno debe estar seguro de que la inversión valdrá la pena.*"

El trabajo se organiza como sigue. En la Sección 2 se dan algunos antecedentes sobre el cambio de arquitectura de procesamiento. En la Sección 3 se dan detalles sobre la metodología propuesta, lo cual incluye los elementos que constituyen el *hardware* del sistema y la forma de aplicar la nueva arquitectura. En la Sección 4 se detallan las pruebas que permiten comparar el funcionamiento de la nueva programación con aquella obtenida con la arquitectura anterior. Finalmente, en la Sección 5 se presentan las conclusiones y el trabajo futuro.

2. ANTECEDENTES

2.1 Arquitectura Blackboard

La arquitectura Blackboard (Matamoros , 2013), se caracteriza por tener una estructura centralizada (Figura **2**), en donde los módulos contienen soluciones parciales y específicas para resolver una tarea administrada por un sólo *maestro*.

Hasta el momento, no se había propuesto ni aplicado alguna arquitectura que no fuera similar a ésta dentro del robot, y es por eso que, si alguno de los módulos dejaba de funcionar (o incluso el administrador de ellos), simplemente no se lograba ejecutar la marcha bípeda, sin saber exactamente en dónde había ocurrido el problema.

2.2 Arquitectura Peer-to-Peer

Esta arquitectura tiene como propósito descentralizar los recursos distribuidos de tarea dedicada dentro del sistema (los recursos de la arquitectura Blackboard). Se utiliza principalmente para evitar la distinción entre clientes y servidores donde todos se comunican con todos. En otras palabras, cualquier Nodo (programa) puede ser maestro y lograr administrar las tareas que se le indiquen (Figura **3**).

Además de tener la capacidad de solucionar el problema presentado en el sistema anterior, este nuevo esquema

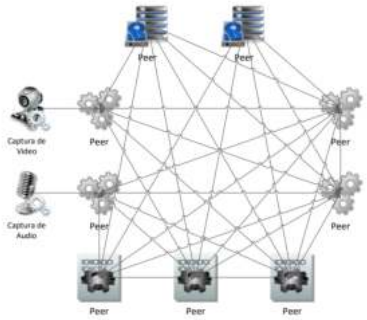

Figura 3. Arquitectura decentralizada Peer-to-Peer (Matamoros , 2013)

brinda la flexibilidad al usuario de aprovechar los datos transferidos entre todos los Nodos, utilizándolos como enlaces para enrutar paquetes de información aún cuando alguno de éstos se interrumpa o se sature. A pesar de parecer una situación más compleja, amplía la perspectiva de soluciones, permitiendo la escalabilidad a nuevas posibilidades de incorporación de módulos, reduciendo costos y reestructurando todo a un sistema modular. ROS funciona precisamente con la arquitectura Peer-to-Peer.

2.3 Protocolos y otras herramientas

Para lograr desarrollar el planteamiento del proyecto, fue necesario tener claro el conjunto de utilidades como lo son el sistema operativo Linux, los protocolos RS232 (serial) e I2C, los sockets, algunos lenguajes de programación (Python, XML y C++), Energia IDE (similar a Arduino) y el protocolo XML/RCP que hace posible el funcionamiento de la arquitectura Peer-to-Peer en ROS mediante los patrones de cliente/servidor o publicador/suscriptor.

2.4 Estructura conceptual de ROS

ROS fue originalmente desarrollado en 2007 por el Stanford Artificial Intelligence Laboratory con el soporte del proyecto Stanford AI Robot. En 2008 se continuó el desarrollo en Willow Garage, laboratorio de investigación robótica dedicada a la creación de software de código abierto. En 2013, ROS se transfirió a la Open Source Robotics Foundation para ser liberado bajo los términos de la licencia de Berkeley Softawre Distribution y un software de libre acceso, gratuito para el uso comercial y de investigación.

Para entender el funcionamiento del nuevo sistema hay que conceptualizar a ROS, desde los protocolos, hasta las convenciones y paqueterías que maneja (ROS Wiki , 2018).

La Figura **4** muestra tres niveles principales: Nivel de Sistema de Archivos, Nivel de Grafo de Procesos y Nivel de Comunidad.

El Nivel de Sistema de Archivos se encarga de todo el ordenamiento de información y programas para que ROS pueda gestionar adecuadamente los procesos.

El Nivel de Grafo de Procesos tiene como propósito relacionar las variables, comunicar y enlazar los conceptos de Tópicos, Servicios, Nodos y Maestro, entre otros. En otras palabras, es la parte funcional de ROS.

El Nivel de Comunidad es un conjunto de repositorios digitales e información útil para el usuario como forma de guía y soporte técnico de otros desarrolladores.

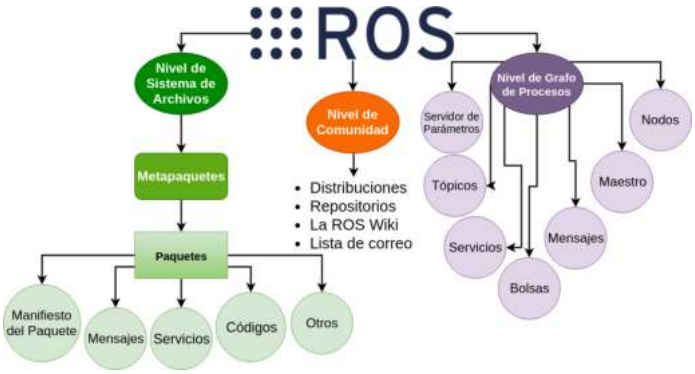

Figura 4. Estructura interna de ROS

Conceptos como Nodos, Tópicos, Mensajes y el Maestro, fueron elementales para este proyecto. Los Nodos no son más que programas que el usuario crea en Python o C++ (por ejemplo) y son configurados como publicadores o suscriptores en este caso. Un Nodo es un archivo ejecutable dentro de un Paquete de ROS. Los Tópicos son aquellas vías de transmisión de datos que se encuentran configuradas dentro de los Nodos e intercambian Mensajes. Un Mensaje es una estructura de datos compuestos y comprende una combinación de tipos primitivos y mensajes, valga la redundancia. Cada Tópico contiene un tipo específico de dato que depende del tipo de sensor que se esté utilizando y de la convención del programador. Estos provienen de las dependencias std msgs, geometry msgs, entre otras. Algunos ejemplos de datos son: Float32MultiArray, Int32, String, Point, etc.

El ROS MASTER proporciona el registro de nombre para todos los Nodos y consulta el resto del grafo de procesos para interactuar con todos los demás elementos. Sin embargo, si deja de funcionar, los demás Nodos pueden seguir interactuando entre sí gracias a la nueva arquitectura.

Cabe señalar que, teniendo en cuenta algunas especificaciones técnicas de las distribuciones (versiones) y herramientas más actualizadas y compatibles a la fecha, se determinó que la implementación de ROS Kinetic Kame era la mejor opción para este robot.

2.5 Ejes de inclinación y dirección

Los Sistemas de Referencia de Orientación y Rumbo (AHRS, IMU + procesador) son dedicados a obtener datos con mayor fiabilidad y precisión que únicamente las Centrales de Mediciones Inerciales (IMU) de los tres ángulos en el espacio. Los AHRS se enfocan en proporcionar los tres ejes (Pitch, Roll y Yaw). Así que se propuso obtener el ángulo Yaw con el dispositivo MPU-6050 ya que el UM7 presentaba variaciones no deseadas en este eje. Sin embargo, se mantuvo la obtención de Roll y Pitch con el UM7 ya que la caminata desarrollada en (Sánchez , 2017) era funcional y no era estrictamente necesario crear una redundancia en los mismos valores para una verificar dichos datos.

3. METODOLOGÍA Y DESARROLLO

La metodología propuesta implica el desarrollo de diagrama de Nodos y Tópicos; le revisión de la instrumentación y eventual diseño de material adicional; la simulación del sistema en una computadora personal; por último, revisar

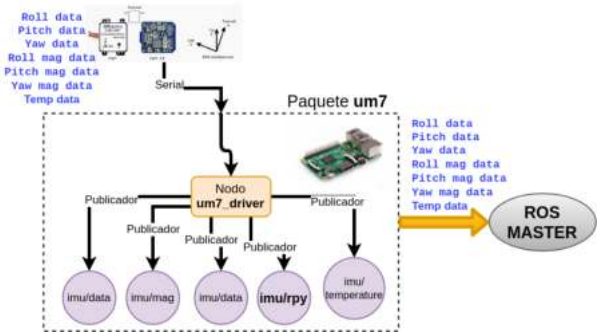

Figura 5. Nodo del AHRS UM7 con cinco Tópicos utilizables dispuestos para el ROS MASTER

los preparativos finales para la puesta en marcha del sistema a bordo del robot. El desarrollo de cada etapa para el caso del robot bípedo se detalla a continuación. Cabe señalar que se procuró aprovechar todo el hardware que se tenía disponible desde trabajos anteriores para adaptarlo a la nueva arquitectura.

3.1 Diagrama de nodos y tópicos

La implementación de Nodos y Tópicos es uno de los elementos más importantes en este desarroll. oFue necesario categorizar y asignar las distintas variables de los sensores en uso del robot que se transportan por medio de los Tópicos a cada uno de los Nodos que las aprovechan especialmente para cálculos de control. El desarrollo de cada Nodo fue distinto en cada módulo debido a su funcionamiento interno o compatibilidad con el sistema, pero con una estructura similar.

Utilizando como ejemplo al Nodo de el AHRS UM7 del la Figura **5** se puede apreciar cómo se encapsulan los datos entregados por vía serial y se gestionan por el Nodo llamado **um7_driver**, el cual publica al Tópico **imu/rpy** y entrega datos de inclinación Roll, Pitch y Yaw. Se enlaza con el ROS MASTER y con esto es posible llamarlo desde otro Nodo en cualquier momento. Afortunadamente éste debía ser únicamente configurado, no programado, a diferencia de los demás (como los de control y obtención de datos de los sensores).

En total se programaron cuatro Nodos incluyendo el anterior: el Nodo de adquisición de datos (**sensores_node**), el Nodo de cálculo y control de postura y marcha (**control_postura_node** y **control_completo_node**) y el Nodo de escritura a servomotores (**tiva_servos_node**).

3.2 Instrumentación y diseño

Se realizó un análisis de especificaciones técnicas del sistema con el que se contaba para concluir si satisfacían las necesidades del sistema descentralizado. Se reutilizaron la mayoría de los componentes y se agregó un par mas, commo un Arduino UNO y el AHRS MPU-6050:

- Raspberry Pi 3B: en donde se instalaría ROS. Es el sistema embebido con la CPU que gestiona todos los procesos de mayor complejidad. Aquí se ubica el ROS MASTER. En ésta se encuentra el Nodo **sensores_node** y **um7_driver**.
- Tiva-C TM4C123G: driver con arquitectura ARM Cortex M4F, es una plataforma de prototipos en una

familia de microcontroladores. En ésta se encuentra el Nodo **tiva_servos_node**.

- Arduino UNO: a pesar de ser una plataforma de prototipado al igual que la Tiva-C, contiene las características necesarias para la adquisición de datos de los sensores de presión y los potenciómetros (Sánchez , 2017). Aquí se encuentra el Nodo **sensores_node**.
- UM7-lt y MPU-6050: el AHRS UM7 obtenía los ángulos Roll y Pitch (Sánchez , 2017), mientras que el MPU-6050 se propuso para obtener el ángulo Yaw (dirección). A pesar de tener cierta deriva en este último ángulo, debido a las distancias recorridas por el robot y el tiempo de las pruebas no es un problema significativo para esta aplicación.

Como material adicional se creó una nueva PCB para el robot que permitiera tener las condiciones necesarias para el funcionamiento de ROS en el robot.

3.3 Simulación del sistema

Debido a que ROS y las bibliotecas para el control del robot suelen instalarse y ejecutarse de manera más rápida y con menos errores en una laptop o computadora de escritorio que en la Raspberry, se optó por simular los Nodos desde Ubuntu 16.04.04 LTS en equipos con procesador Core i5 e i7 antes de configurarlos dentro de los dispositivos.

3.4 Preparativos finales

Una vez verificados los Nodos y Tópicos con pruebas de escritorio, se procedió a configurar los parámetros de la imagen de Ubuntu Mate de la Raspberry Pi3, programar los Nodos dentro de la Tiva y el Arduino, y crear los scripts y el archivo roslaunch para el empaquetamiento de procesos.

4. PRUEBAS Y ANÁLISIS DE RESULTADOS

La naturaleza de este robot hace que su funcionamiento sea lento. Diversos factores como el par proporcionado por los actuadores, las configuraciones de transmisión de datos, la calidad de la red y el tiempo de procesamiento de los programas de control definen un tiempo de muestreo cercano al típico para este tipo de aplicaciones. Las pruebas mostradas en https://youtu.be/Wf6W14zqIGw determinaron que es necesaria la frecuencia de publicación de cada Nodo de por lo menos 22 Hz obtenida con la herramienta **rostopic hz**, lo que equivale a un tiempo de muestreo de 45 milisegundos. Este tiempo de muestreo puede reducirse realizando un análisis de todo el sistema tomando en cuenta el desempeño de cada dispositivo, la cantidad y los tipos de datos transmitidos, lo cual no se consideró en los alcances de este trabajo.

Diversas aplicaciones en tiempo no real pueden aplicarse con ROS, desde monitorear los datos de una casa, hasta robots de rescate para su intervención en desastres naturales (Koubaa , 2016) cuyas frecuencias de muestreo típicas se encuentran en un rango promedio de entre 10 y 30 Hz.

4.1 Comportamiento del control de postura

El experimento del control de postura se basó en localizar al robot sobre una plataforma plana inclinándola hasta 15° aproximadamente en 2 segundos como se muestra en

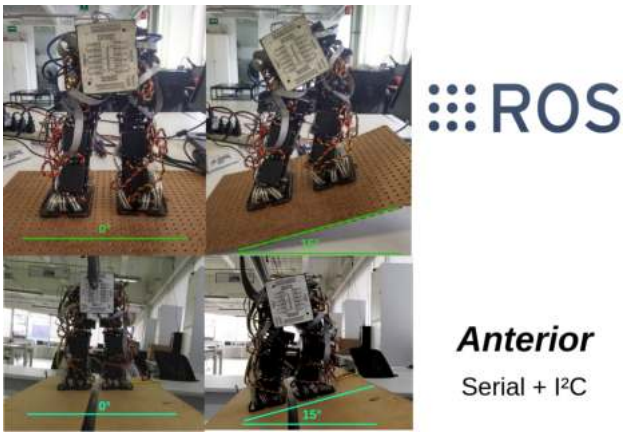

Figura 6. Experimento de inclinación a 15° y corrección de postura con ambos sistemas (Ricárdez , 2018) (Sánchez , 2017) en 2 segundos

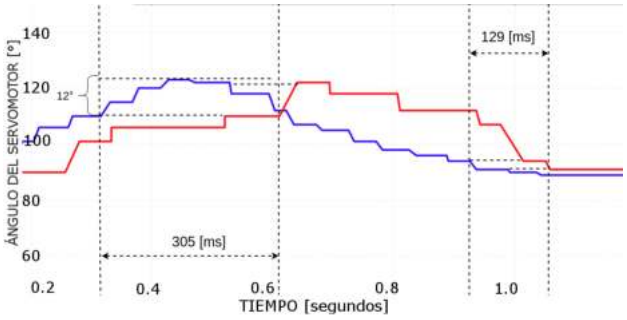

Figura 7. Control de postura de inclinación izquierda en 1 segundo. Referencia (azul) vs. respuesta (roja)

la Figura **6**. El comportamiento en el sistema previo y el nuevo sistema con ROS fue similar. Ambos sistemas corrigieron la postura en función de las inclinaciones proporcionadas por el AHRS UM7.

Sin embargo, en ambos sistemas se presentó un retraso en la respuesta, siendo éste cercano a los 100 milisegundos en el sistema anterior y 200 milisegundos con ROS. Era evidente después de varias muestras que el uso del nuevo sistema y su interacción con elementos ya antes mencionados amplificaban dicho retraso. Con el fin de dimensionarlo, se realizó un experimento de inclinaciones súbitas para simular la situación en la que el robot tuviera que corregir su postura con mayor rapidez (de caída o con una marcha más veloz). Se utilizó la herramienta **rqt_plot** para graficar los datos de actuación de un servomotor del robot. Las Figura **7** y **8** muestran un retraso de 305, 129 y 269 milisegundos, con un error absoluto de hasta 12°, siendo la gráfica azul lo que tenía que realizar el robot, y la roja lo que realmente efectuó.

4.2 Comportamiento del control de marcha

Al igual que con el experimento del control de postura, se realizó el mismo experimento en superficie plana e inclinada a 8° con el control de marcha en ambos sistemas (Figura **9**). A diferencia de la prueba de inclinaciones súbitas, en esta ocasión las respuestas entre ambos sistemas fueron prácticamente idénticas ante el ojo humano, así que se propuso graficar de la misma forma con rqt_plot las instrucciones hacia el robot y la respuesta proveniente de él

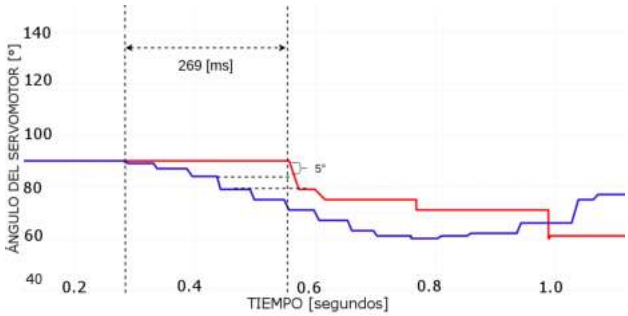

Figura 8. Control de postura de inclinación derecha en 1 segundo. Referencia (azul) vs. respuesta (roja)

Figura 9. Comparación de respuesta en superficie plana e inclinada de ambos sistemas (Ricárdez , 2018) (Sánchez , 2017)

para registrar de manera cuantitativa las diferencias entre ambas arquitecturas. La Figura **10** muestra un error menor en los ángulos del robot debido a que la marcha bípeda es mucho más lenta en comparación del experimento de inclinaciones súbitas en el control de postura. La explicación de por qué se realizó primero el caso de la postura fija es porque era complicado determinar con exactitud lo que sucedía a nivel computacional entre ambas curvas usando sólo la marcha bípeda. Pero ahora que se sabe que el error absoluto máximo que se registró fueron 2° y 130 milisegundos de retraso en la marcha bípeda, se puede confirmar que el sistema con los mismos controladores es funcional con ROS. El robot no se cayó en ninguna de las pruebas finales. Tomando en cuenta que a partir del 14.86 % el robot comenzaba a presentar irregularidades en su caminata, el 70 % de las veces alcanzó valores por debajo de este umbral. Dicho umbral se consiguió realizando un experimento con retraso intencional, aumentándolo gradualmente para encontrar el valor al que el robot ya no conseguía realizar una caminata adecuada.

4.3 Procesamiento

Se utilizó la herramienta Scout Realtime (Scout Realtime , 2018) para monitorear las métricas de Linux en tiempo real. Análogo a la herramienta top, SR permite visualizar de manera gráfica el consumo de procesamiento del robot, siendo definido por su página oficial como "el top del desarrollador moderno".

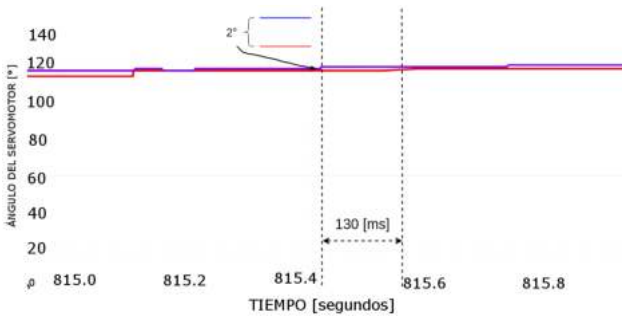

Figura 10. Control de marcha en 1 segundo. Referencia (azul) vs. respuesta (roja)

Tabla 1. Consumo de procesamiento y memoria **con ROS**

Proceso de IMG 16GB	**CPU Raspberry Pi3**	**MEM Raspberry Pi3**	**CPU HP Pavilion 15**	**MEM HP Pavilion 15**
Ninguno	**3 %**	**28 %**	7.19 %	17 %
roslaunch: maestro y nodos	**22.9 %**	**34 %**	7.19 %	17 %
Control Postura	**22.9 %**	**34 %**	22.92 %	18 %
Control Marcha	**22.9 %**	**34 %**	15.51 %	19 %

Tabla 2. Consumo de procesamiento y memoria **sin ROS**

Proceso ejecutado - IMG 64GB	**CPU Raspberry Pi3**	**MEM Raspberry Pi3**
Ninguno	**4 %**	**13 %**
Control de postura	**37.7 %**	**17 %**
Control de marcha	**44.7 %**	**18 %**

Las Tablas **1** y **2** muestran a ambos sistemas ejecutando distintos procesos y se graficaron en la Figura **11**. Se observó que con ROS existe un menor uso de CPU y un mayor consumo de MEM, mientras que en el sistema anterior ocurre exactamente lo contrario. Una de las mayores ventajas que se observó con el nuevo sistema fue que el mayor procesamiento se realizaba por medio de objetos remotos. En otras palabras, después de invocar al maestro y a los Nodos del robot, los demás procesos de control se podían realizar desde un equipo externo, de manera mucho más rápida y más segura. Es por esta misma razón que, a diferencia del sistema anterior que va aumentando gradualmente con cada proceso su consumo de CPU y MEM, ROS evita alcanzar los límites de aprovechamiento de las métricas de la Raspberry y permite al robot seguir funcionando de manera eficiente. Además, es más adecuado generalmente tener en rangos bajos el uso de CPU que de MEM, como es este caso.

4.4 Estabilidad computacional

Definiendo la estabilidad computacional como requerimiento del sistema de cómputo de no presentar fallas importantes a lo largo de toda la marcha bípeda, el sistema con la arquitectura anterior presentó múltiples de ellas en varias ocasiones a la hora de ejecutar los programas principales en la terminal de procesos de Ubuntu, además

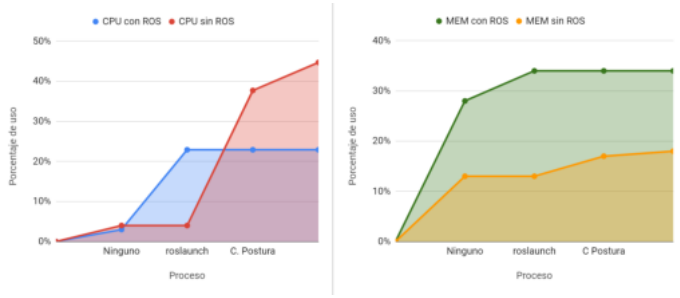

Figura 11. Comparación de procesamiento. CPU: ROS (azul) vs. anterior (roja) y MEM: ROS (verde) vs. anterior (amarilla)

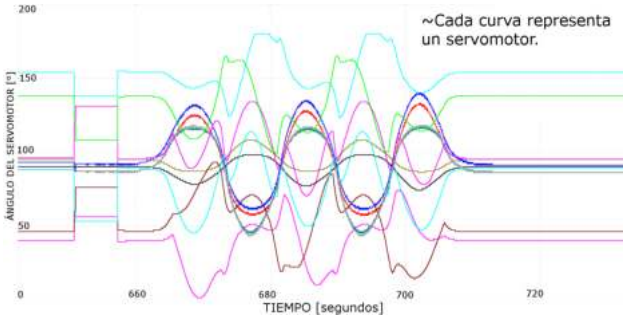

Figura 12. Evolución de la posición de los 12 servomotores durante la marcha bípeda en 63 segundos.

de ocasionar detención repentina del robot a media caminata. No obstante, con ayuda de la herramienta rqt_plot de ROS, se pudieron graficar los datos variables de los doce servomotores en 63 segundos (que es la duración de la marcha bípeda) como se muestra en la Figura **12**. No se observó ninguna curva discontinua o mal funcionamiento en las pruebas de la caminata bípeda en el nuevo sistema.

4.5 Implementación de ambos AHRS

Desde un principio, el AHRS UM7 presentaba señales no deseadas en el eje Yaw y Azimuth. El que presentó menor dispersión en sus datos estáticamente fue Azimuth, pero presentó un error relativo del 12.5 % (Figura **13**). En cambio, al aplicar el eje Yaw del MPU-6050 en el robot, se observó que se mantuvo exactamente en la misma posición gracias al filtro complementario en toda su curva por el trabajo colaborativo del acelerómetro y giroscopio. Se notaron algunas variaciones en los datos del MPU-6050 con la herramienta rostopic echo en el orden de centésimas, estimándose un error relativo menor a 0.1 %, considerándose éste despreciable para el robot.

El conjunto de los tres ejes, Roll, Pitch y Yaw, no presentó variaciones notorias a lo largo de su graficación. Con la ayuda de la herramienta rostopic echo se registraron errores relativos menores a 0.1 %, representando éste una magnitud de falla despreciable para el robot. Ahora se sabe que los dos primeros ejes del UM7 (Roll y Pitch) funcionan correctamente para las leyes de control de postura y marcha, y que el eje del MPU-6050 (Yaw) sirve para alguna futura ley de control para caminatas en curvaturas.

5. CONCLUSIONES Y TRABAJO A FUTURO

Se lograron modularizar y unificar por completo los procesos de control anteriormente desarrollados para la postura

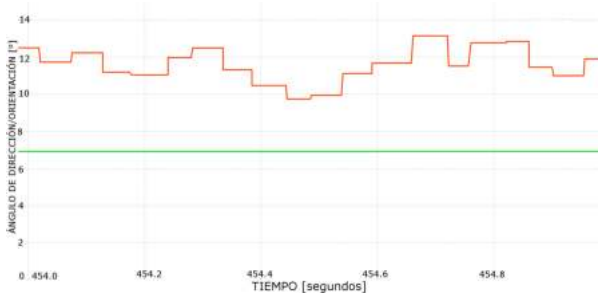

Figura 13. Comparación de comportamiento estático entre Azimuth UM7 (naranja) y Yaw MPU-6050 (verde)

y marcha bípeda, lo cual mejoró significativamente la estabilidad computacional de sus procesos. A pesar de que en este caso ROS incrementa el retraso en la respuesta, el funcionamiento del control de marcha de este robot sigue lográndose debido a su dinámica lenta.

El precio que se pagó al implementar este sistema en particular fue un mayor consumo de memoria y las dificultades de programación propias de la migración a esta nueva arquitectura, pero se confirma que ROS es una alternativa viable dentro de la implantación de sistemas de control en robots.

Como trabajo futuro se realizará un análisis que permita seleccionar mejores dispositivos (hardware) con el fin de lograr reducir el tiempo de muestreo y mejorar el desempeño del robot, hasta lo que permitan los actuadores que se tienen actualmente.

Asimismo, se prevé la implantación de otros controladores bajo esta nueva arquitectura, por ejemplo un control de ZMP, así como su funcionamiento en paralelo con otras leyes de control, complementándose y ofreciendo un mejor desempeño.

REFERENCIAS

A. Sánchez (2017). *Controladores difusos para postura y marcha de un robot bípedo de 12 GDL.* Tesis, Universidad Nacional Autónoma de México, Ciudad de México.

J. Matamoros (2013). *Análisis de extensibilidad, reestructuración y desempeño de software para robots móviles.* Tesis, Univ. Nal. Autónoma de México, Cd. de México.

A. Ricárdez (2018). *Modularización y unificación de sistemas para el control de un robot bípedo con ROS.* Tesis, Univ. Nal. Autónoma de México, Cd. de México.

ROS Wiki (2018). *Documentation.* Consultado en Junio del 2018. Recuperado de: http://wiki.ros.org/

Boston Dynamics (2018). *Robots.* Consultado en Junio del 2018. Recuperado de: https://www.bostondynamics.com/robots

BipedRobin (2015). *Institute for Robotics at Johannes Kepler University Linz.* Consultado en Junio del 2018. Recuperado de: http://wiki.ros.org/Robots/BipedRobin

A. Koubaa (2016). *Robot Operating System The Complete Reference Vol. 1.* Springer

Scout Realtime (2018). *Top for the modern developer.* Consultado en Noviembre del 2018. Recuperado de: https://scoutapp.github.io/scout_realtime/

J. O'Kane (2017). *A Gentle Introduction to ROS.* Artículo, Universidad del Sur de California, pp. 2-3.

B. Siciliano, O. Khatib (2008). *Springer Handbook of Robotics.* Springer-Verlag.

