UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Modularización y unificación de sistemas para el control de un robot bípedo con ROS

TESIS

Que para obtener el título de

Ingeniero Mecatrónico

P R E S E N T A

Adrián Ricárdez Ortigosa

DIRECTOR DE TESIS

Dr. Edmundo Gabriel Rocha Cózatl

Ciudad Universitaria, Cd. Mx., 2019

Dedicado a mis padres Jorge Alberto Ricárdez Santos y Evert Ileana Ortigosa Dorantes,
y a mi hermano Jorge Alberto Ricárdez Ortigosa, quienes siempre estuvieron para apoyarme.
Saben que éste es solo el principio de un gran proyecto que tengo por delante,
ya que me prometí usar el conocimiento de la ingeniería siempre por el bien de la humanidad.

Agradecimientos

Agradezco al excelente apoyo brindado por el M.I. Marco Antonio Negrete Villanueva con la impartición del curso particular de ROS para otros cuatro compañeros y yo. También, por su completa disposición en las asesorías de los días requeridos.

A mi compañero Julio César Oceguera González, que actualmente está cursando su maestría, por haber colaborado conmigo en el desarrollo del proyecto y siempre tener la voluntad de realizar labores de investigación e implementación de conocimiento.

Quiero hacerle un gran reconocimiento a mi colega Ing. Allen Eduardo Sánchez Ortega por estar al tanto de las diversas preguntas y propuestas que se generaron a lo largo del proyecto, ya que él fue uno de los tesistas anteriores inmediatos al comienzo del mío.

A la Facultad de Ingeniería de la UNAM y a todos mis profesores y maestros, por todo el tiempo y conocimiento dedicado a guiarme para ser el gran futuro ingeniero que siempre quise ser.

Al Programa de Vinculación con los Egresados de la UNAM y la Dirección General de Orientación y Atención Educativa por el apoyo brindado durante la investigación y desarrollo de este trabajo.

Y por supuesto, agradezco enormemente al Dr. Edmundo Rocha Cózatl por su honestidad y profesionalismo, siempre dispuesto a dar su mejor esfuerzo como docente y administrativo de su carrera.

Índice general

Índice de figuras

Índice de tablas

Resumen

En este trabajo se implementó un nuevo sistema con arquitectura *Peer-to-Peer* en un robot bípedo de doce GDL (Grados de Libertad) con objeto de unificar y modularizar los procesos para el control de postura y marcha. Para lograr dicho propósito, se realizó un listado de dispositivos en función de la comparación entre el actual sistema y el previo a él. Esta transformación fue llevada a cabo por medio de ROS (Robot Operating System), el cual es un conjunto de bibliotecas de software y herramientas que ayudan al usuario a crear aplicaciones de robots. La hipótesis se enfocó en la posibilidad de incrementar la estabilidad de los procesos computacionales del robot y mejorar la eficiencia de trabajo para futuros proyectos por medio de esta herramienta de código abierto.

Teniendo establecida la nueva estructura de trabajo, se programaron los códigos llamados Nodos, los cuales transmiten y reciben información de variables obtenidas de la lectura de sensores a través de patrones de paso de mensajes llamados Tópicos, que sirven como medio de transporte para las variables de cálculo utilizadas en los controladores de postura y marcha bípeda.

Junto con un mantenimiento mecánico preventivo y correctivo, se diseñó y manufacturó una nueva Tarjeta de Circuito Impreso para facilitar las conexiones internas del robot, disminuyendo considerablemente las fallas electrónicas del sistema.

Finalmente, se configuraron todos los parámetros necesarios del sistema y se realizaron los análisis pertinentes a los resultados de las pruebas de corrección de postura, marcha bípeda y, por último, un Nodo que realiza navegación autónoma como comprobación de la hipótesis abordada.

Este trabajo se caracteriza por tener respaldo y soporte completo de todo el sistema con ROS, permitiendo a otros desarrolladores continuar con la investigación e implementar nuevos controladores en el robot.

Capítulo 1

Introducción

1.1. Estado del arte

1.1.1. ROS (Robot Operating System)

La página oficial de ROS [3] menciona:

"*Robot Operating System (ROS) es un conjunto de bibliotecas de software y herramientas que ayudan al usuario a crear aplicaciones de robots. Desde los controladores hasta los algoritmos de última generación y con potentes herramientas de desarrollo, ROS tiene lo que necesitas para tu próximo proyecto de robótica. Y es todo de código abierto.*"

El objetivo de ROS es simplificar el trabajo de manipular el comportamiento de un robot complejo a través de una diversidad de módulos y plataformas estandarizadas para robótica. Además de ser ampliamente utilizado en la robótica, ROS tiene un gran campo de aplicación en la ingeniería.

Según la referencia de una página de Acutronic Robotics [22], 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, incubadora de empresas y laboratorio de investigación robótica dedicada a la creación de software de código abierto para las aplicaciones para robots personales con más de 20 instituciones colaborando en el desarrollo. En febrero de 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. Esto promovió la reutilización de programas, lo que facilitó a los desarrolladores y científicos aprovechar las herramientas informáticas, trabajando

con repositorios de distintos países.

¿Por qué se creó ROS?

Las pequeñas y grandes empresas desde siempre han tenido el problema al crear y manipular un software efectivo para cada plataforma que se desarrolla día a día. Además, la tecnología es una de las cosas más cambiantes hoy en día debido a la acelerada evolución de la sociedad tecnológica.

Se puede analogar un ejemplo muy sencillo:
La primer industria de computadoras aplica Outsourcing para terciarizar a otra empresa dedicada a la fabricación de tornillos para sus productos, pero, ¿qué sucede?. Resulta que la carcasa de la computadora (el producto) fue construida por sus mejores ingenieros y tiene una alta eficiencia a nivel de diseño para manufactura y ensamble. El problema es que los agujeros que se diseñaron para el ingreso de dichos tornillos no soportan los que existen en el mercado de su país y es muy caro comprarlos por importación. La primera industria le exije a la segunda que manufacture tornillos especiales puesto que ya se tienen las piezas base, pero la segunda se reúsa, a menos de que las compren al triple del precio. Los primeros acceden al pago. Pasan unos días, se cierra el trato y los ingenieros de la segunda empresa comienzan a trabajar, pero se dan cuenta de que las geometrías del tornillo requerido, así como la cuerda, son imposibles de calcular con su actual software. Se manda llamar a otra empresa para que les presten su licencia temporal para finalizar ese trabajo. Consiguen terciarizar temporalmente a una empresa de software de alto nivel, pero pierden mucho dinero en la licencia y la segunda empresa termina tablas con sus ganancias. Con el tiempo encima, terminan el trabajo, pero se percatan de que las computadoras de la compañía no logran exportar el tipo de archivo que se genera del programa de la tercera empresa. Como resultado, la primera no consigue sus tornillos y no vende sus computadoras, la segunda quiebra y la tercera no recibe completo su pago.

Esta es una historia hipotética que es difícil que ocurra, ya que actualmente en las empresas existe un rígido protocolo en la revisión de contratos y supuestamente buena comunicación. Aún con todo esto existen empresas grandes que quiebran por errores absurdos. Algunos de sus problemas son la estandarización, falta de investigación y comunicación.

ROS, a pesar de llevar poco más de 8 años, resuelve muchos de los problemas mencionados. Estandariza los protocolos de comunicación, establece convenciones entre programas y permite visualizar, controlar y sincronizar los procesos casi en tiempo real.

Las *distribuciones* (versiones) de ROS actualmente disponibles son:

1. ROS Box Turtle
2. ROS C Turtle
3. ROS Diamondback
4. ROS Electric Emys
5. ROS Fuerte Turtle
6. ROS Groovy Galapagos
7. ROS Hydro Medusa
8. ROS Indigo Igloo
9. ROS Jade Turtle
10. **ROS Kinetic Frame** (ésta es la de interés)
11. ROS Lunar Loggerhead
12. ROS Melodic Morenia

La ROS Wiki [3] informa la programación de cada versión que libera al público, siendo ésta aproximádamente en Mayo de cada año. Cuando se realiza un lanzamiento en año par, la *distribución* adquiere como característica ser de Soporte de Larga Duración (LTS, por sus siglas en inglés). Las razones por las cuales se eligió **Kinetic Frame** fueron la siguientes:

- Su lanzamiento ocurrió en 2016, siendo ésta una de las más recientes con un Tiempo de Vida (EOL, por sus siglas en inglés) de 5 años, con un soporte hasta 2021.
- Es compatible con una de las versiones de Ubuntu más recientes (aparte de 18.04 LTS Bionic Beaver [26-04-2018]), la cual es Ubuntu 16.04 LTS Xenial Xerus, lanzada el 21 de abril de 2016 con un soporte hasta 2021 también.
- La información de la ROS Wiki del 3 de Junio de 2018 menciona que, además de ser preferida por sus paqueterías, actualizaciones y soporte, es altamente recomendada por sus desarrolladores.
- Tiene todas las herramientas necesarias para este proyecto. La versión Indigo también cumple este requisito, pero su EOL es en 2019, además de que la comunidad web se encuentra más activa con **Kinetic**.

- Al inicio de esta tesis, **Kinetic** era la más nueva. Sin embargo, a lo largo del proyecto fue lanzada la versión Melodic (2018), pero ya se tenía un avance e investigación desarrollada sobre la penúltima versión, considerándose **Kinetic** como semi-nueva.

1.1.2. Robótica alrededor del mundo

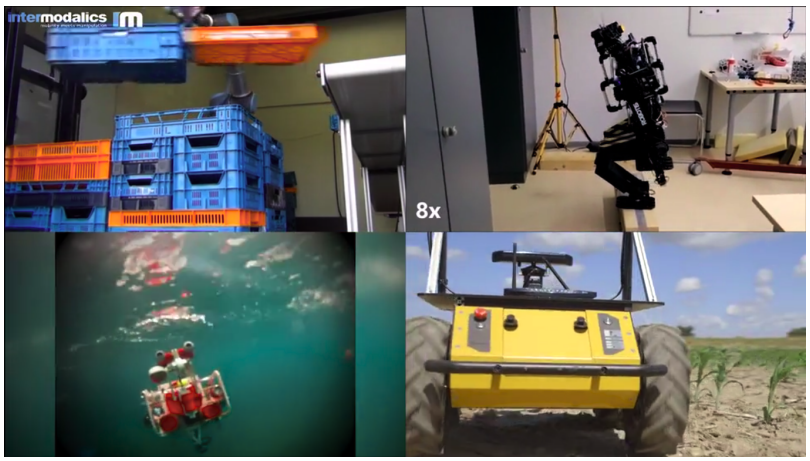

Figura 1.1: Robots de tareas industriales implementando ROS [3]

Figura 1.2: Robots en acción autónoma con ROS [3]

ROS tiene su mayor auge en la robótica moderna y comenzó a implantarse en los *cerebros* inteligentes de los robots en muchas universidades y centros de investigación.

Los robots de las Figuras **1.1**, **1.2** y **1.3** tienen distinas áreas de aplicación, como lo son la navegación autónoma, automatización industrial, reconocimiento de patrones e investigación.

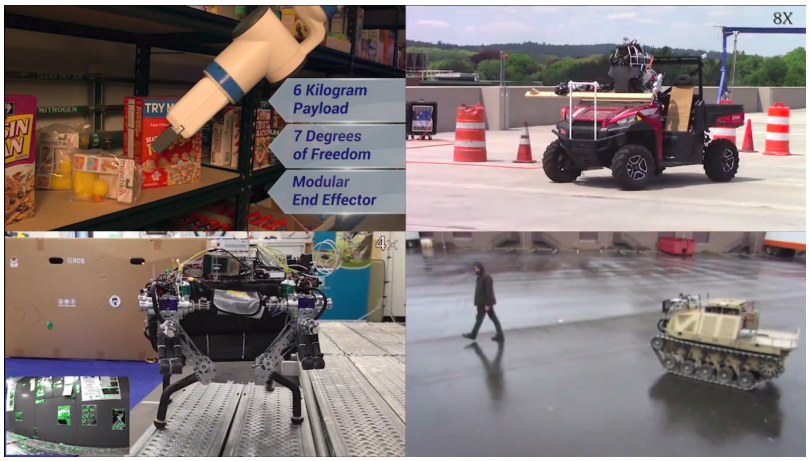

Figura 1.3: Robots con reconocimiento dimensional con ROS [3]

Entre ellos se encuentran el famoso BigDog (cuadrúpedo) y Atlas (humanoide) creados por Boston Dynamics®[13], los cuales se desarrollaron con ROS.

1.1.3. Robot bípedo con ROS

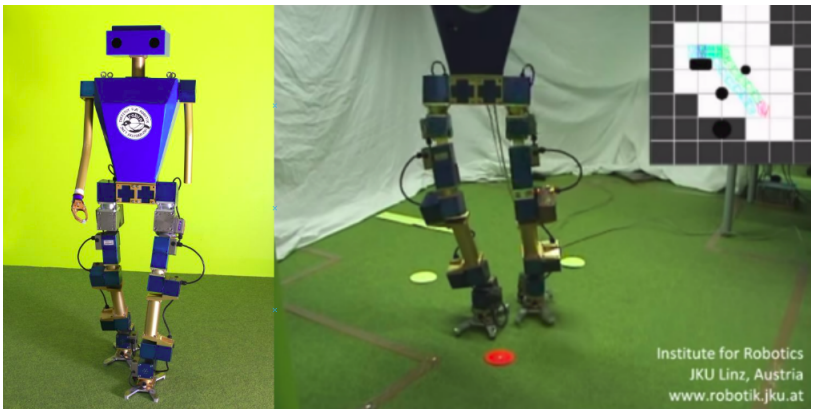

Figura 1.4: BipedRobin del Institute for Robotics at Johannes Kepler University Linz [14]

Actualmente, estos robots se desarrollan como proyectos de investigación y algunos de fabricación múltiple para lanzamientos en el mercado. Son usados como objeto de apoyo al ser humano en sus labores diarias o específicas. Cada vez se sabe más de ellos gracias a las publicaciones de trabajos profesionales, experimentos dentro de laboratorios, difusión en revistas científicas y, sobre todo, mediante las redes sociales.

Particularmente hablando de los robots bípedos y humanoides, se pueden enlistar algunas de sus tantas aplicaciones:

- Prototipos para desarrollo e investigación
- Modelos del cuerpo humano en medicina
- Apoyo doméstico
- Apoyo en misiones espaciales
- Seguridad
- Entretenimiento (animatrónica)
- Industria: empaquetamiento y control de calidad

Uno de los artículos que más llamó la atención acerca de la implementación en un robot bípedo fue BipedRobin [14] en el año 2015, un robot bípedo desarrollado por el Institute for Robotics at Johannes Kepler University Linz. El proyecto de la Figura 1.4 utilizó Ubuntu 10.04 LTS, RTAI, y ROS Fuerte para su ejecución. Esto confirma que es posible hacer uso de ROS para objetivos particulares, como lo es un robot bípedo.

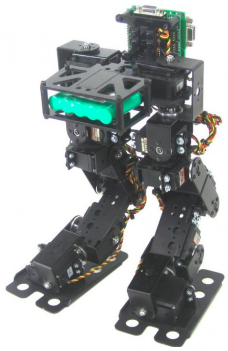

Figura 1.5: Lynxmotion Biped Robot Scout®[4]

1.2. Planteamiento del problema

En el año 2008, el Departamento de Mecatrónica de la Facultad de Ingeniería de la UNAM adquirió el robot bípedo Scout (Figura 1.5) con la finalidad de que la institución desarrollara herramientas físico-matemáticas, electrónicas y programables para implementar un control de marcha de dicho robot a través del Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica (PAPIIT).

Los trabajos anteriores más relevantes que definieron el estado del actual robot fueron realizados por:

- Ulises Jiménez [16] en el modelo carro-mesa para la obtención de trayectorias de marcha en línea recta del robot bípedo.
- Saddán Hernández [2] con la renovación y el mejoramiento de actuadores e instrumentación.

- Fernanda Merino [5] con el modelado en lazo abierto simplificado del carro-mesa parametrizado para la obtención de trayectorias de marcha en línea recta, arco de circunferencia, y subida no constante del centro de masa.
- Y, recientemente, Allen Sánchez [1] con el control difuso para postura y marcha bípeda del actual robot.

A pesar de los grandes logros que se obtuvieron, comenzaron a presentarse una serie de problemas que deterioran y entorpecen el estado del robot, junto con las acciones de caminata respectivamente. Algunos de estos errores son acumulativos, lo que significa que no se han podido resolver desde hace varios años atrás y las consecuencias han empezado a manifestarse. Otros son espontáneos, los cuales se generan por causas desconocidas o van de la mano con el tipo de sistema que se tiene. Los problemas son los siguientes:

- Desincronización entre programas de marcha bípeda y datos de la instrumentación: *checksum error* y ruptura de bus de datos con la Unidad de Medidas Inerciales (IMU, por sus siglas en inglés) modelo UM7
- Desconexión repentina con los datos de la instrumentación
- Deterioro mecánico en articulaciones y eslabones, incluyendo aflojamiento de tornillería y pequeñas deformaciones en piezas
- Detención espontánea de marcha bípeda sin saber el problema ni la solución
- Ruido extremo en los datos de dirección (Yaw) y orientación del magnetómetro (Azimuth) de la UM7
- Daño permanente en tarjeta de adquisición de datos de potenciómetros y Flex, Teensy 3.1
- Falsos contactos en electrónica

Además, en el trabajo a futuro de la tesis de Allen [1] se menciona la posible unificación de controladores (Figura **1.6**). Sin embargo, el programar un nuevo controlador no es el mayor problema, sino que **no existe** una plataforma con suficiente soporte para que los demás desarrolladores de licenciatura y maestría puedan seguir mejorando el proyecto de manera eficiente. Esto incluye la independización de procesos, una guía rápida de soluciones, control de versiones, y una plataforma de monitoreo visual de la información acerca de lo que ocurre en el robot en el instante de la operación. Estabilizar el sistema significa disminuir los fallos computacionales por debajo de un

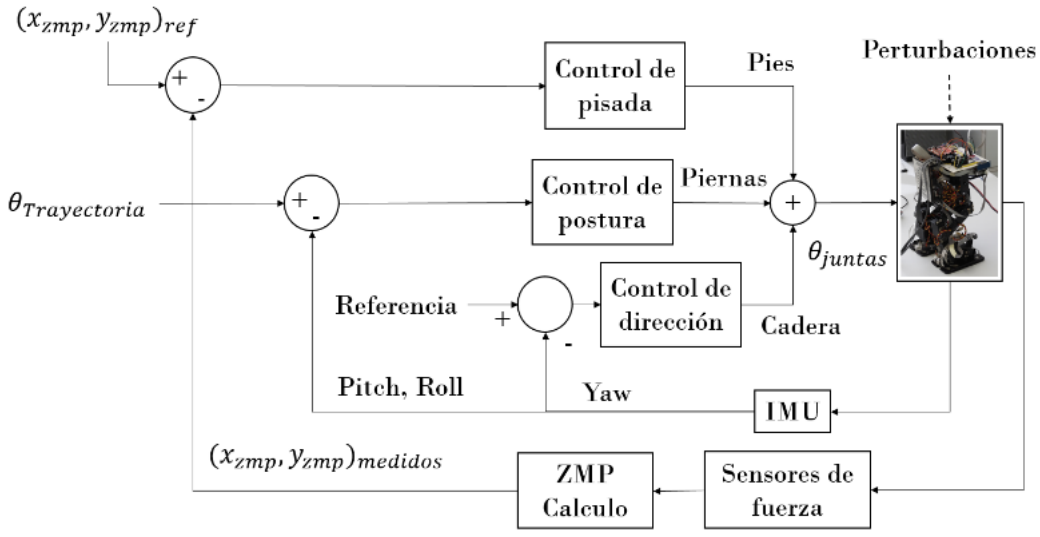

Figura 1.6: Diagrama de bloques propuesto para el robot bípedo Scout [1]

umbral, el cual varía dependiendo del nivel de estabilidad requerido en el sistema [9].

En este contexto, la presente tesis abordará el siguiente tema: **modularización y unificación de sistemas para el control del robot**, ya que surge de la necesidad de migrar éstos a una nueva plataforma o realizar cambios importantes de hardware y software en el robot, esperando solucionar la mayoría o todos los problemas mencionados anteriormente.

1.3. Hipótesis

Se puede incrementar considerablemente la estabilidad de los procesos computacionales del robot bípedo en ejecución del control de postura y de marcha por medio de ROS. A partir de ello, es posible crear las bases de una nueva estructura de trabajo enfocada a la modularización de sistemas para un futuro desarrollo más eficiente, en este caso, siendo eficiencia toda aquella metodología capaz de realizarse y cumplir adecuadamente su objetivo en el menor tiempo posible.

1.4. Justificación

¿Por qué ROS es la mejor opción?

De ahora en adelante, el concepto de *sistema anterior* será referenciado a toda la estructura de software y hardware desarrollada en las tesis inmediatas anteriores.

Debido a que se utilizan protocolos de comunicación combinados (I^2C y RS232) entre los distintos dispositivos, no se han logrado detectar aún los momentos exactos de la pérdida de transferencia y recepción de datos, ni mucho menos las causas (que pueden ser variadas).

Es imperativo poner atención y dar soluciones a los problemas actuales para un inmediato seguimiento de los proyectos, en beneficio del Departamento y de los estudiantes.

Se comprobó que el control difuso fue buena estrategia y profesionales del área han sugerido que el robot es una excelente herramienta para la investigación y desarrollo con aplicaciones en ingeniería.

Analizando algunas de las configuraciones posibles, se contempló la migración del sistema embebido Raspberry Pi3 Modelo B (elegida en la tesis de Allen [1]) a uno con mayor capacidad de procesamiento y memoria. La tarjeta BeagleBone Black y PcDuino3 (PC + Arduino) eran las candidatas, donde la Beaglebone es ligeramente más potente que la Raspberry en algunos aspectos y la PcDuino más potente que las otras dos.

¿Es el procesamiento realmente el problema?

Observando las características técnicas detenidamente de la Raspberry Pi Modelo B [6], se llegó a la conclusión de que realiza un trabajo excelente a pesar de no ser tan poderosa como una computadora ordinaria. En realidad, lo que hace falta es monitorear los procesos del robot y las interacciones entre sus distintos tipos de variables al momento de la ejecución, ya que, al observar el comportamiento de estas variables, se puede realizar un diagnóstico más acertado acerca de las fallas presentadas. Entre las configuraciones restantes del sistema (como reemplazo de módulos/sensores, adición de memoria RAM, entre otras) se decidió por la mejor candidata que resolvería prácticamente todos los problemas actuales. Dicha configuración se muestra en la Tabla **1.1**.

Además, la Tabla **1.2** presenta un resumen de las tecnologías investigadas en la tesis de José Matamoros [46] *Análisis de Extensibilidad, Reestructuración y Desempeño de Software para Robots Móviles*, siendo ROS una de las mejores opciones de herramienta de desarrollo.

1.5. Objetivo general

Modularizar y unificar los procesos de control anteriormente desarrollados en el robot bípedo Scout para el control de postura y marcha con ROS.

Tabla 1.1: Configuración de dispositivos electrónicos del robot bípedo

Elemento nuevo	Ventajas	Desventajas
Robot Operating System	Modularización de procesos, soporte de plataforma, comunidad y foros altamente activos, licencia abierta, entorno colaborativo con control de versiones, estandarización con Linux, independización de programas, integración con proyectos de código abierto [8].	Posibles incompatibilidades, instalación tardada, ocasionales fallas en archivos de configuración de ROS.
IMU UM7-LT	Filtro Kalman integrado en uno de los mejores sensores inerciales del mercado.	Datos erróneos de dirección y orientación.
IMU MPU-6050	Bajo costo, muy comercial, bastante soporte en internet.	Mayor número de elementos a la lista de módulos.
Tiva-C TM4C123G	Existe un módulo activo en el proyecto, sencillez en programación (como un Arduino), excelente velocidad de ejecución a 80MHz (entre otras características), independización como módulo de actuación.	Problemas de comunicación con el protocolo I^2C.
Arduino UNO + Multiplexores	Microcontrolador con velocidad y hardware suficiente para obtención de datos de los sensores Flex, potenciómetros e IMU simultáneamente a 16MHz. Fácil programación.	Pequeña adición de tamaño y peso en la electrónica del robot con respecto al módulo anterior (Teensy 3.1).

Tabla 1.2: Tecnologías de software para desarrollo de robots móviles de la tesis [46]

	Arquitectura	Plataforma	Lenguajes	Intercambio de información	Protocolo de Comunicación
CARMEN	Centralizada Cliente/Servidor	Linux	C/C++, C#, Java, PHP, Python, Ruby	Paso de mensajes Variable Compartida	IPC
MS RDS	Distribuida Cliente/Servidor	Windows	C#, Lenguaje Visual	Paso de Mensajes	WCF
MIRO	Distribuida *Peer-to-Peer*	Linux, OS X, Windows	C/C++, Java, Python (Soporte de CORBA)	Objetos remotos	TAO CORBA
MOOS	Centralizada *Blackboard*	Linux, OS X	C++	Variable compartida	Mensaje MOOS
OpenRDK	Centralizada *Blackboard*	Linux, OS X	C/C++	Variable compartida	Mensaje OpenRDK
Orca	Distribuida *Peer-to-Peer*	Linux, OS X, Windows	C++, C#, Java, PHP, Python, Ruby (Soporte de ICE)	Objetos remotos	ICE
Player / Stage	Distribuida *Peer-to-Peer*	Linux, OS X, Windows	C/C++, C#, Java, PHP, Python, Ruby	Paso de mensajes	Mensaje Player
SPQR-RDK	Distribuida Cliente/Servidor	Linux, OS X	C/C++	Variable compartida	Mensaje SPQR-RDK
ROS	Distribuida *Peer-to-Peer*	Linux, OS X, Windows	C/C++, C#, Java, PHP, Python, Ruby	Objetos remotos	XML/RPC

1.6. Objetivos particulares

- Evaluar cualitativamente (por medio de observación) el nuevo sistema y compararlo con el anterior.
- Evaluar cuantitativamente (por medio del comportamiento de las variables a través del tiempo) el nuevo sistema y compararlo con el anterior.
- Dar mantenimiento electrónico y mecánico para disminuir fallas ajenas al objetivo de control de marcha.
- Brindar soporte completo del proyecto para futuros desarrollos.
- Tener control total sobre los procesos que se ejecutan en el robot con ROS.
- Proponer una solución para el eje de dirección/orientación del robot.
- Determinar si ROS es una herramienta adecuada para el nuevo sistema.

1.7. Alcances y limitaciones

Alcances

- El proyecto tendrá soporte completo (repositorio de archivos, respaldo de imagen virtual y guía de pasos de ejecución).
- Todos los alcances de los trabajos anteriores se mantendrán y potenciarán para realizar mejoras en el futuro.
- Se realizará un Nodo de comprobación de desarrollo en ROS como ejemplo de implementación de un nuevo controlador, utilizando el dato de dirección/orientación para navegación autónoma.

Limitaciones

- La lectura de los sensores Flex para el ZMP (Zero Moment Point) y potenciómetros todavía no estarán incluídos en el control de marcha.
- Las limitaciones de inclinación en el control de postura y marcha se conservarán.
- Se realizará una posible actualización de software hasta el 2021 a ROS **Melodic Morenia**.

Capítulo 2

Conceptos preliminares

Este capítulo menciona los conceptos y requerimientos básicos para desarrollar nuevas ideas en el robot bípedo. Para ello se tiene que estar familiarizado con algunos comandos de Ubuntu (Apéndice **N**) y con los lenguajes de programación Python y C++, así como sus respectivas extensiones y compiladores.

Requerimientos en la computadora:

- Instalar ROS **Kinetic** full-desktop y sus paqueterías en Ubuntu 16.04.4 LTS desde la página oficial [3].
- Confirmar que se tienen instaladas las últimas versiones de Python (con 2.7 y 3.6 basta).
- Instalar la IDE de Energia para manejar el ambiente de desarrollo de la tarjeta controladora Tiva-C, basada en la IDE de Arduino, desde su página oficial [19].

Otras herramientas útiles:

- Instalar Terminator, el cual ayuda a tener una mejor visualización de los procesos, ya que divide en multiterminales una sola ventana [27].
- Tener instalado el editor de texto enriquecido SublimeText [25].

2.1. Linux - Ubuntu 16.04.4 LTS (Xenial Xerus)

GNU/LINUX es un Sistema Operativo (SO) como MacOS y Windows (Figura **2.1**), definido como el software que administra los recursos de hardware de la computadora. [26].

Figura 2.1: Sistemas operativos más utilizados [15]

Se eligió a Ubuntu 16.04.4 LTS como SO ya que soporta ROS **Kinetic**. Además, se encuentra bastante soporte de ambos en la comunidad web.

2.2. Protocolos de comunicación

2.2.1. RS232

Introducido por Electronic Industries Association en 1960, el protocolo RS232 (Serial) es un estándar a nivel mundial que rige sobre los parámetros de comunicación por medio de las líneas Transmisión (Tx) y Recepción (Rx). Antes de hacer uso de él es necesario establecer la velocidad de envío y recepción de los paquetes de datos, llamado Baudrate, los niveles de voltajes utilizados (3.3v o 5v), el tipo de cable permitido, distancias entre equipos, etc. [10].

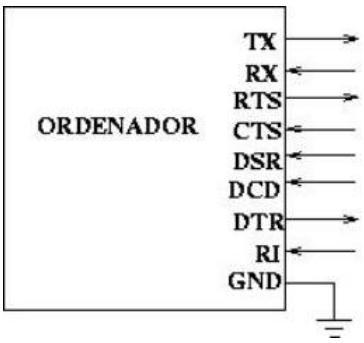

Figura 2.2: Protocolo serial RS232 [10]

La mayoría de las computadoras y módulos electrónicos, desde básicos hasta complejos, suelen tener un puerto disponible para este tipo de comunicación. Todos los puntos de operación de la norma son: Request to Send, Clear to Send, Data Terminnal Ready, Data Set Ready, Recieve Signal Line Detect, Transmit Data, Recieve Data.

2.2.2. I²C

Uno de los problemas que se observó con los años fue que los puertos seriales son asíncronos. Esto significa que no tienen un dato de reloj en ejecución independiente si existe transmisión o recepción de datos, haciéndolo totalmente propenso a ciertos errores si alguno de los dos falla por sólo un poco. Además, el puerto serial requiere un hardware donde se tenga una sincronía y calidad excelentes. Esto es como decirle a una persona que comience a hablar exactamente en el momento en el que el otro termina de hacerlo sin previo aviso.

Posteriormente, se inventó el protocolo serial SPI, pero requería de un pin extra por cada esclavo que se tuviera.

En 1982, Philips introdujo el protocolo Inter-Integrated Circuit (I²C) en sus chips que, aunque no era tan potente como el actual (tan sólo 112 dispositivos en aquel entonces y con limitaciones de direcciones), logró dar un salto enorme en la electrónica y telecomunicaciones [11].

El hardware del I²C es más complejo que el SPI, pero menos que el RS232, además de que soluciona el problema del reloj como en la Figura 2.3. Para entender de una forma más sencilla, se ejemplifica un debate presidencial: Hay un reloj tomando el tiempo de habla por cada uno de los candidatos, ya que es visto que si hay más de un político hablando y atacándose contra otro, las propuestas deseadas para el público se tiran por la borda y no se resuelve nada.

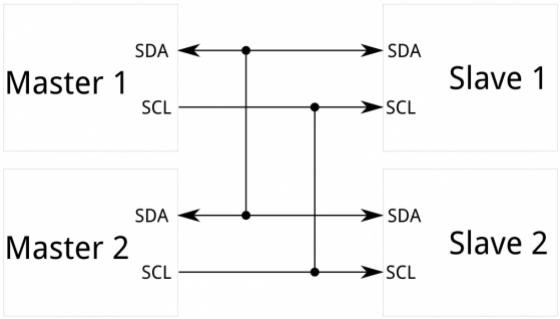

Figura 2.3: Protocolo I²C [11]

La mayoría de los dispositivos I²C responden entre 100[kHz] y 400[kHz], donde existe una condición inicial, dirección de búsqueda de dispositivo, datos y condición de detención. Todo esto lo hace por medio de combinación de señales cuadradas y decifrado de direcciones en lenguaje hexadecimal por cada dispositivo [11].

2.2.3. Sockets

Son una forma de comunicación entre computadoras. ROS ya tiene patrones de comunicación incluidos que los configuran automáticamente, simplificando el trabajo. Estos patrones son TCPROS, que es una capa de transporte de datos. Utiliza conectores TCP/IP (Transmission Control Protocol/Internet Protocol) estándar, en los cuales las entradas se reciben a través de un socket de servidor TCP con un encabezado que contiene al Mensaje y la información de enrutamiento [24].

2.2.4. XML/RPC

Se trata de un protocolo servidor-cliente que utiliza como lenguaje para su codificación XML y HTTP (Puerto 80) como medio de transmisión, normalmente abierto. Es multilenguaje (PHP, python, java, c++, etc.), de software libre y simple de utilizar. Fue creado en 1998 por Dave Winder de Userland Software y Microsoft [12].

Los pasos para la ejecución son los siguientes:

- El cliente inicia el proceso con una Llamada a Procedimiento Remoto (RPC, por sus siglas en inglés) enviando una petición a un servidor conocido. En la petición le debe indicar al servidor qué procedimiento se debe ejecutar para suministrarle los parámetros necesarios (si se requieren, dependiendo del servicio).
- El servidor envía una respuesta al cliente y la aplicación continúa su ejecución. En el caso de ROS, puede ser 1:1 (un cliente para un servidor, bloqueante) o 1:n (uno a n Nodos, no bloqueante).

2.3. Arquitectura *Peer-to-Peer* (Igual a Igual) vs. arquitectura *BlackBoard*

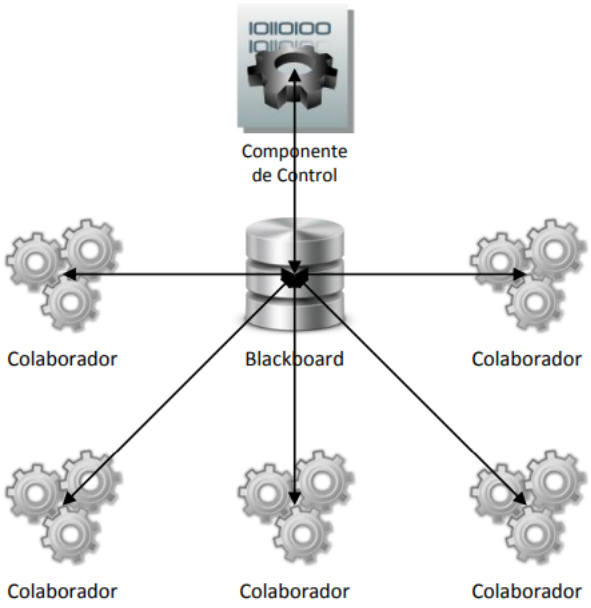

Figura 2.4: Arquitectura *BlackBoard* [46]

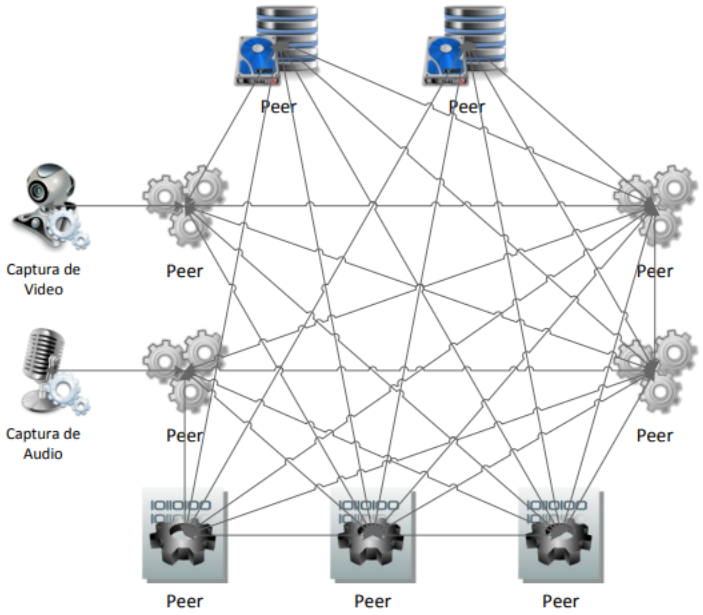

Figura 2.5: Arquitectura *Peer-to-Peer* [46]

La arquitectura *Peer-to-Peer* [7] se encarga de descentralizar los recursos distribuidos en un sistema. Se utiliza principalmente para evitar la distinción entre clientes y servidores donde todos se comunican con todos, a diferencia de la arquitectura *BlackBoard* [46], la cual es centralizada y es similar a la arquitectura que anteriormente ha sido utilizada en el robot bípedo. La arquitectura *BlackBoard* se encarga de solucionar un problema mediante el conjunto de soluciones parciales de módulos especializados a dicha tarea (Figura **2.4**).

Esto no quiere decir que la estructura *Peer-to-Peer* no lo haga, sino que, además de tener la capacidad de solucionar el problema presentado, ésta 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 más compleja, amplía la perspectiva de soluciones, permitiendo la escalabilidad a nuvas posibilidades de incorporación de módulos, reduciendo costos y reestructurando todo el sistema [46]. **ROS funciona precisamente con la arquitectura *Peer-to-Peer***.

2.4. Lenguajes de programación

2.4.1. Python

Es un lenguaje de scripting independiente de plataforma que soporta herencia y polimorfismo preparado para realizar cualquier tipo de programa, desde aplicaciones de Windows a servidores de red o incluso, páginas web. Esto significa que es un lenguaje interpretado que no necesita ser compilado para ejecutarse, lo que ofrece ventajas como la rapidez de desarrollo [17]. Sin embargo, esto tiene un costo, ya que al ser lenguaje de alto nivel y reducir sus líneas de código, también aumenta el tiempo de ejecución en algunas ocasiones.

En los últimos años el lenguaje se ha hecho muy popular, gracias a varias razones:

- La cantidad de bibliotecas que contiene, tipos de datos y funciones incorporadas en el propio lenguaje que ayudan a realizar muchas tareas habituales sin necesidad de tener que programarlas desde cero.
- La sencillez y velocidad con la que se crean los programas. Un programa en Python puede tener de tres a cinco líneas de código menos que su equivalente en Java o C.
- La cantidad de plataformas en las que se puede desarrollar, como Unix, Windows, OS/2, Mac, Amiga y otros.
- El código abierto de Python que permite el desarrollo a nivel empresarial.

Un programa de Python 2.7 o 3.6 puede ser utilizado con el comando en la terminal:

python + *nombre-del-programa*.**py**

Pero en este proyecto no fue necesario utilizarlo, ya que ROS ejecuta los programas con sus propios comandos estándar de paquetes.

2.4.2. C++

Este es un lenguaje de programación orientado a objetos (al igual que el anterior) que toma la base del lenguaje C. Fue diseñado a mediados de los 80 y la intención de su creación fue extender al exitoso lenguaje de programación C con mecanismos que permitieran la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, C++ es un lenguaje híbrido [18].

2.4.3. Energia IDE

Arduino IDE y Energia IDE son plataformas similares con los mismos principios y fundamentos, pero con colores (Figura **2.6**), controladores y microcontroladores diferentes. La instalación de Energia contiene soporte para la tarjeta MSP430. Sin embargo, es posible agregar nuevos dispositivos como la Tiva-C (la que se va a utilizar), CC3200, MSP432, por medio del administrador de tarjetas en Tools–Board–Board Manager.

Figura 2.6: Energia IDE basada en Arduino [30]

2.5. Conceptos básicos de ROS

A pesar de tener las palabras de Sistema Operativo en su nombre, ROS no forma parte de ellos. Es un conjunto de paquetes y herramientas de software (a diferencia de Linux, que sí es un SO) para desarrollar aplicaciones en robótica principalmente.

ROS tiene tres niveles conceptuales: el nivel del Sistema de Archivos, el nivel del Grafo de Procesos, y el nivel de Comunidad, los cuales se pueden consultar a mayor detalle en [20] y [21]. Algunos de los comandos más usados para el manejo del robot se encuentran en el Apéndice **N**.

2.5.1. Nivel de Sistema de Archivos

- **Paquetes**: Son la unidad principal para organizar software en ROS. Puede contener procesos (Nodos en ROS), una biblioteca dependiente de ROS, conjuntos de datos, archivos de configuración o cualquier otra cosa que sea útil organizar y conjuntar más archivos.
- **Metapaquetes**: Son Paquetes especializados que sólo sirven para representar grupos relacionados de paquetes de información.
- **Manifiesto del Paquete (package.xml)**: Proporciona metadatos acerca del Paquete. Los metadatos son aquellos que describen el contenido de los archivos o de la información de los mismos (nombre del Paquete, versión, descripción, información de licencia y dependencias).

- **Repositorios**: Son una colección de Paquetes que comparten un Sistema de Control de Versiones (como GitHub) común.

- **Tipos de Mensaje (msg)**: Definen las estructuras de datos para los Mensajes enviados en ROS.

- **Tipos de Servicio (srv)**: Definen las estructuras de datos de solicitud y respuesta para los Servicios en ROS.

2.5.2. Nivel del Grafo de Procesos

Como bien ya se mencionó en la Sección **2.3**, el grafo de procesos es la red *Peer-to-Peer* de los procesos de ROS que interactúan con datos en forma conjunta. En el caso del robot bípedo son datos de sensores y cálculos de los controladores.

Los conceptos básicos del grafo de procesos son: ROS MASTER, Nodos, Servidor de Parámetros, Mensajes, Servicios, Tópicos y Bolsas. Cada uno propociona datos al grafo de procesos de diferentes maneras. Más adelante, en la Sección **2.5.8**, se mencionan algunas de ellas.

2.5.3. Nivel de Comunidad

- **Distribuciones**: Son colecciones en la pila de versiones que se pueden instalar en el equipo. Afortunadamente, éstas son consistentes con las distribuciones de Linux.

- **Repositorios**: ROS se basa en una red federada (unida) de repositorios de códigos, donde las diferentes instituciones pueden desarrollar y lanzar sus propios componentes de software de robots.

- **La ROS Wiki**: La comunidad de la ROS Wiki de la página principal sirve para documentar información acerca de ROS. Cualquiera puede registrarse para obtener una cuenta y contribuir con su propia documentación, proporcionar actualizaciones y correcciones, escribir tutoriales, etc. Además, tiene su propio foro de preguntas y respuestas [45] y un blog informativo [44].

- **Lista de correo**: Es el canal de comunicación principal sobre las nuevas actualizaciones de ROS, así como un foro para realizar preguntas.

2.5.4. Paquetes y Dependencias

Los Paquetes son carpetas de archivos que contienen por defecto dos de ellos elementales:

- **CMakelists.txt**: Contiene instrucciones para la compilación de Paquetes para que puedan funcionar correctamente los Nodos.
- **package.xml**: Contiene todos los metadatos, los cuales describen el contenido del Paquete de ROS, describiendo su contenido, calidad, condiciones, versiones, disponibilidad, creador, etc.

Cualquier carpeta que contenga estos dos mencionados, es un Paquete. Éste puede contener archivos que definan Mensajes y/o Servicios.

Para crear un Paquete se hace uso del comando **catkin_create_pkg [nombre-del-paquete] [dependencias]**. A la hora de crear el Paquete, hay que hacer referencia a las dependencias que el Nodo utilizará. En el caso del robot bípedo se hará uso de: **std_msgs, roscpp, rospy, sensor_msgs y geometry_msgs**.

2.5.5. ROS MASTER

El ROS MASTER proporciona el registro de nombre para todos los Nodos y consulta el resto del grafo de procesos como se muestra en la Figura **2.7**. Sin el ROS MASTER, los Nodos no serían capaces de encontrar al resto de ellos, intercambiar Mensajes o siquiera invocar Servicios. Ejecutar este proceso requiere el comando **roscore**, que es lo primero que se tiene que hacer para lograr invocar los Nodos y visualizar los Tópicos activos. La única excepción es si se ejecuta un archivo **.launch** que realiza todo esto automáticamente (Sección **2.5.12**).

2.5.6. Servidor de Parámetros

Permite almacenar datos mediante una clave en una localización central, la cual actualmente es parte del ROS MASTER. Es un diccionario compartido multivariable accesible a través de la red. Los Nodos usan el Servidor de Parámetros para almacenar y recibir parámetros en tiempo de ejecución. ROS tiene una herramienta para trabajar con dicho Servidor llamada **rosparam**:

- **rosparam list**: Lista todos los parámteros del Servidor.
- **rosparam get** *PARÁMETRO*: Devuelve el valor de un parámetro.
- **rosparam set** *PARÁMETRO VALOR*: Establece el valor de un parámetro.

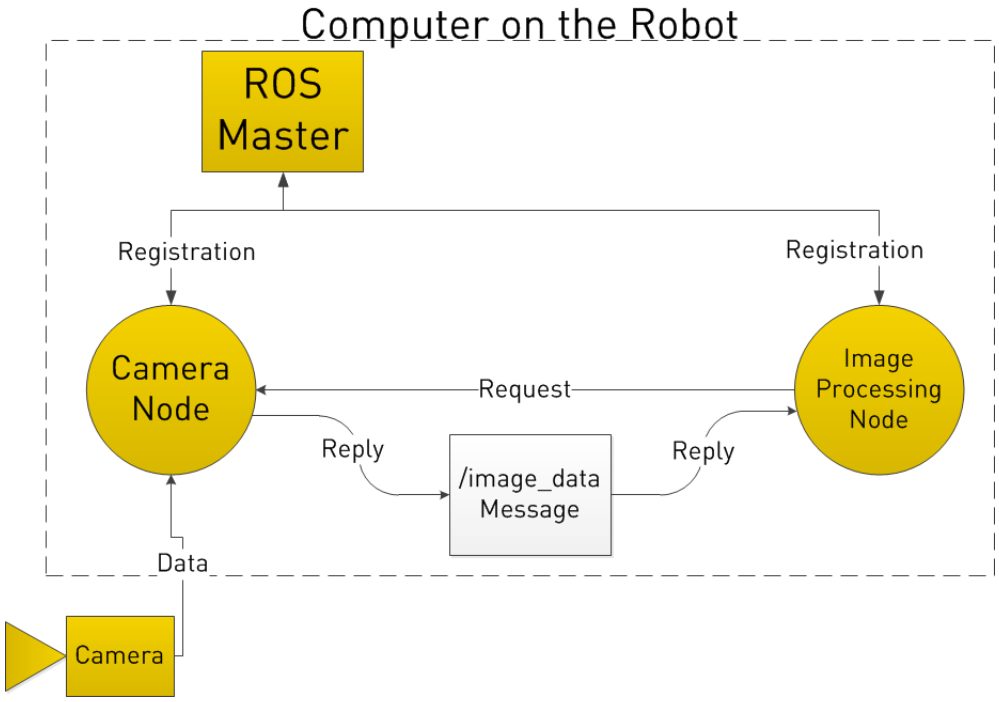

Figura 2.7: Ejemplo de funcionamiento de ROS por medio de intercambio de datos en un Tópico y dos Nodos [23]

- **rosparam delete** *PARÁMETRO*: Elimina un parámetro.
- **rosparam dump** *ARCHIVO*: Guarda los parámetros del Servidor en un fichero.
- **rosparam load** *ARCHIVO*: Carga un fichero con parámetros en el Servidor de Parámetros.

Para crear, modificar y trabajar con Paquetes, ROS proporciona algunas herramientas de asistencia:

- **rospack**: Se usa para obtener información o buscar información sobre un Paquete.
- **roscreate-pkg**: Crea un nuevo Paquete.
- **rosmake**: Compila un Paquete.
- **rxdeps**: Ve las dependencias de un Paquete como un grafo.

Y para moverse entre los diversos Paquetes y ficheros, ROS proporciona uno llamado **rosbash**, que proporciona algunos comandos similares a los de Linux:

- **roscd**: Ayuda a encontrar y moverse de directorio rápidamente.

- **rosed**: Edita un archivo.
- **roscp**: Copia un fichero de algún Paquete.
- **rosd**: Lista los directorios de un Paquete.
- **rosls**: Lista los ficheros de un Paquete.

2.5.7. Nodos

Un Nodo es un archivo ejecutable dentro de un Paquete de ROS. Utiliza una biblioteca cliente de ROS para comunicarse con otros Nodos. Pueden publicar o suscribirse a un Tópico, el cual es configurado directamente desde el código.

Un Nodo de ROS funciona con el uso de las bibliotecas cliente de ROS, roscpp (C++) y rospy (Python).

Para poder ver la lista de Nodos activos se utiliza el comando **rosnode list**, donde siempre estará activo **/rosout**, el cual siempre se ejecuta a medida que recoge y registra la salida de depuración de los Nodos.

Para poder ejecutar un Nodo, una vez compilado desde el espacio de trabajo (*workspace*) con el comando **catkin_make**, se hace uso de **rosrun** *NOMBRE-DEL-PAQUETE NOMBRE-DEL-NODO*.

El Nodo se puede crear simplemente abriendo un archivo de texto en un editor (como SublimeText o Geany) guardándolo con la extensión correspondiente (.cpp o .py) dentro del Paquete en la carpeta src. Si se utiliza C++, se necesita seguir la convención de nombre como el Paquete creado + "_node".

ROS tiene herramientas para manejar los Nodos y dar información acerca de ellos, como lo es **rosnode**:

- **rosnode info** *NODO*: Muestra información sobre el Nodo.
- **rosnode kill** *NODO*: Mata el Nodo o envía una señal para matarlo.
- **rosnode list**: Muestra una lista con los Nodos activos.
- **rosnode machine** *HOSTNAME*: Muestra la lista de los Nodos ejecutandose en una máquina en concreto.

- **rosnode ping** *NODO*: Muestra la conectividad con el Nodo.
- **rosnode cleanup**: Limpia la información de registro para Nodos inalcanzables.

2.5.8. Mensajes

Los Nodos se comunican mediante el paso de Mensajes. Un Mensaje es una estructura de datos compuestos y comprende una combinación de tipos primitivos y mensajes, valga la redundancia. Los tipos primitivos de datos (integer, floating point, boolean, etc.) están soportados, como los arreglos de datos de tipo primitivo. Los Mensajes pueden incluir estructuras y arreglos (como en las estructuras de C). Cada Tópico contiene un tipo específico de dato que proviene de **std_msgs, geometry_msgs**, entre otros:

- **Float32MultiArray**: multiarreglo de datos flotantes de 32 bits
- **Float64**: dato flotante único de 64 bits
- **Int32**: dato entero único de 32 bits
- **Bool**: dato único booleano (digital, [0,1])
- **Point**: con las mismas propiedades del flotante de 64 bits, pero en posición **X**, **Y** y **Z**
- **String**: paquete de datos de tipo cadena de caracteres

No son los únicos que existen, pero son de los más útiles para lecturas de varios sensores como los Flex y los potenciómetros (**Float32MultiArray**), para lectura de ángulos de algún módulo de inercia (**Float64**), para graficación (**Point**), y otras aplicaciones que se les quiera dar.

2.5.9. Tópicos

Los Tópicos se conocen como buses sobre los cuales los Nodos intercambian Mensajes. Los Tópicos tienen una semántica de publicación/suscripción anónima como en la Figura 2.8. En general, los Nodos no saben con quiénes se están comunicando. En cambio, los Nodos que están interesados en los datos se suscriben al Tópico de interés. Puede haber varios editores y suscriptores de un Tópico. Esto significa que los Tópicos pueden ser transmitidos sin una comunicación directa entre Nodos, lo cual conlleva a una producción y consumo de datos desacoplada como se mencionó en la Figura 2.5.

ROS tiene una herramienta para trabajar con los Tópicos llamada **rostopic** en línea de comandos que proporciona información sobre el Tópico o publica datos directamente sobre la red:

- **rostopic bw** */TOPICO*: Muestra el ancho de banda utilizado por un Tópico.
- **rostopic echo** */TOPICO*: Muestra el Mensaje por la salida estandar.
- **rostopic find** *TIPO-DE-MENSAJE*: Busca Tópicos que usen el tipo de Mensaje especificado.
- **rostopic hz** */TOPICO*: Muestra la tasa de publicación del Tópico.
- **rostopic info** */TOPICO*: Muestra información sobre el Tópico, el Tópico publicado, los que están suscritos y los Servicios.
- **rostopic list**: Muestra una lista de los Tópicos activos.
- **rostopic pub** */TOPICO* **type args**: Publica datos al Tópico. Permite crear y publicar datos en cualquier Tópico directamente desde la línea de comandos (terminal).
- **rostopic type** */TOPICO*: Muestra el tipo de Tópico, es decir, el Mensaje que publica.

En pocas palabras, los Tópicos son la vía por donde se van a transmitir los datos de interés [40].

2.5.10. Servicios

El modelo publicador/suscriptor es un paradigma de comunicación muy flexible, pero en muchos casos el transporte en un único sentido no es suficiente para las interacciones de petición y respuesta que a menudo se requieren en un sistema distribuido como lo es ROS. Un Nodo proporciona un Servicio con un nombre y un cliente utiliza dicho Servicio mediante el envío del mensaje de petición y espera la respuesta.

Cuando se necesita comunicarse con Nodos y recibir una respuesta, no se puede realizar con Tópicos, sino que se utilizan Servicios, los cuales son desarrollados por el usuario. Con **rosservice** se pueden listar y enviar peticiones:

- **rosservice call** */SERVICIO* **args**: Llama al Servicio con los argumentos apropiados.
- **rosservice find** *TIPO-DE-MENSAJE*: Busca los Serivcios por el tipo de Servicio.
- **rosservice info** */SERVICIO*: Muestra información sobre el Servicio.
- **rosservice list**: Lista los Servicios activos.

- **rosservice type** */SERVICIO*: Muestra el tipo de Servicio.
- **rosservice uri** */SERVICIO*: Muestra el Servicio ROSRPC URI.

2.5.11. Patrones

Hay dos tipos de Patrones que se basan en el protocolo XML/RPC (Sección **2.2.4**) en ROS:

- **publicador/suscriptor**: Comunicación por Tópico (analogía con variables globales), 1:n (de 1 a n Nodos) no bloqueante. Eso significa que no espera respuesta de los Nodos donde publica como se muestra en la Figura **2.8**. Este Patrón será el más útil para el robot bípedo.

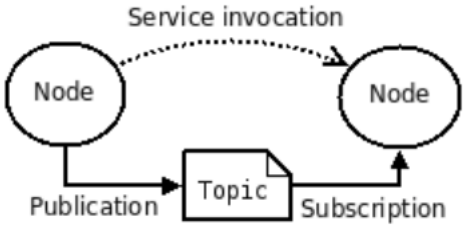

Figura 2.8: Concepto de publicador-suscriptor [21]

- **cliente/servidor**: Comunicación por Servicios (analogía a funciones globales), 1:1 bloqueante. Eso significa que espera respuesta para seguir su proceso.

2.5.12. Archivos Launch y Scripts

Los archivos Launch son muy comunes en ROS tanto para usuarios como para desarrolladores. Proporcionan una forma conveniente de iniciar múltiples Nodos y un maestro (ROS MASTER), así como otros requisitos de inicialización como la configuración de parámetros de cada dispositivo. Éste se encarga de levantar todos los Nodos y Tópicos. También configura el Servidor de Parámetros. Cabe recalcar que una vez que se ejecuta un archivo **.launch**, automáticamente se ejecuta el **roscore** (ROS MASTER).

Estos archivos utilizan un formato XML específico. Se pueden colocar en cualquier lugar dentro de un directorio de Paquetes, pero es común crear un directorio llamado Iniciar o Launch Files dentro del directorio del espacio de trabajo para organizar todos los archivos de inicio [36].

2.5.13. Aprendiendo ROS

Para obtener información práctica y más detallada con ejemplos se recomienda consultar la guía rápida *Aprendiendo_ROS*.pdf que se encuentra en el Repositorio de GitHub en el Apéndice

M, sintetizada para un aprendizaje básico y un rápido entendimiento de los conceptos de ROS y sus aplicaciones.

2.6. Ejes de inclinación: Roll y Pitch

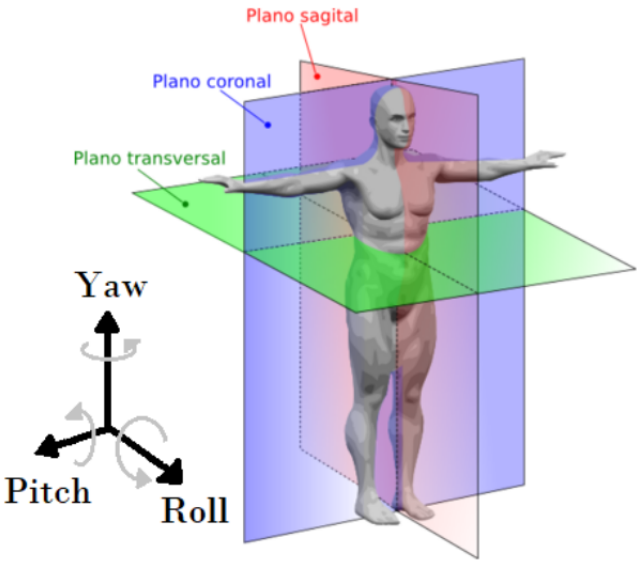

Figura 2.9: Sistema de referencia con los ejes RPY como analogía de una persona con el robot bípedo [1]

En [1] se explica con mayor detalle los planos de referencia y las fases del ciclo de marcha bípeda de la Figura **2.9**.

El funcionamiento de la IMU UM7, dedicada a obtener los tres ángulos de orientación, se enfoca en proporcionar la inclinación frontal y trasera por medio del eje Pitch, las inclinaciones laterales por medio de Roll y el eje de dirección en Yaw. En el control de postura y marcha bípeda sólo se manejan los primeros 2: Roll y Pitch.

2.7. Eje de dirección: Yaw

No debe confundirse el eje de dirección con el eje de orientación, aunque se encuentren en la misma posición.

En navegación autónoma, aeronáutica y otras áreas de la ingeniería que aprovechan los datos de sistemas de referencia, es muy común utilizar el término Azimuth, que es el encargado de brindar un dato en el sistema absoluto terrestre (brújula electrónica). Éste es colineal con el eje Yaw, que también brinda dirección, pero no son iguales. La diferencia entre ellos es que el dato **Azimuth de origen siempre tendrá la referencia hacia el norte de la Tierra**, ya que utiliza un magnetómetro para medir los campos magnéticos del planeta [28], comportándose como una brújula electrónica como lo muestra la Figura **2.10**. El dato **Yaw parte de una referencia relativa para brindar cambios en el giro de ese eje gracias al trabajo colaborativo de un acelerómetro y giroscopio**; es decir, sin tomar en cuenta el norte absoluto.

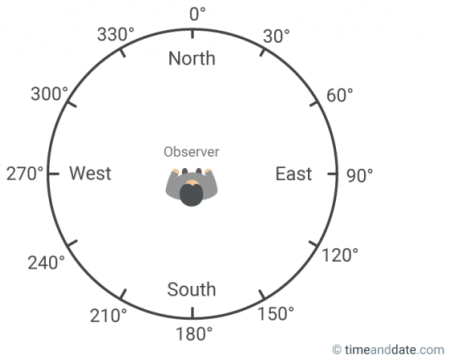

Figura 2.10: Dato Azimuth como analogía a una brújula electrónica

El dato adecuado para el robot en este caso es Yaw, ya que no es de interés realmente ir hacia el norte, sino posicionarlo de tal modo que pueda realizar futuras caminatas con curvaturas dependiendo de la zona en donde se encuentre, tomando como referencia frontal (**90°**) la posición inicial que el usuario le establezca.

Capítulo 3

Descripción del antiguo sistema

Se describirán de manera general los elementos mecánicos, la instrumentación, los diagramas de conexiones y control, las pruebas realizadas y el algoritmo de uso que se ha utilizado para estudiar al robot bípedo.

3.1. Arquitectura mecánica

Diseñado por Lyxmotion®, Scout es un robot de doce grados de libertad con seis articulaciones en cada una de sus dos extremidades. Ambas piernas se unen en un punto en común, siendo éste la cadera, obteniendo trece eslabones en total. La Figura **3.1** muestra la distribución de elementos mecánicos y electromecánicos, siendo de color rojo los actuadores, de negro el esblabon B, que al mismo tiempo se conecta con las uniones amarillo y azul fuerte. Las rodillas están identificadas con azul claro para la parte superior y azul cielo para la parte inferior. Los tobillos se encuentran con café claro y el efector final de color morado (que son los pies [1]).

3.2. Instrumentación

El proyecto cuenta actualmente con una restauración de servomotores e instrumentación [2], ya que a lo largo de más de diez tesis asesoradas, el robot sufrió desgastes inminentes. A continuación, se enlistan los elementos electrónicos que ayudan al control en lazo cerrado [1]:

1. Driver Servomotores - Tiva-C TM4C123GXL ®
2. Módulo IMU UM7-lt ®
3. Sistema embebido Raspberry Pi 3B ®

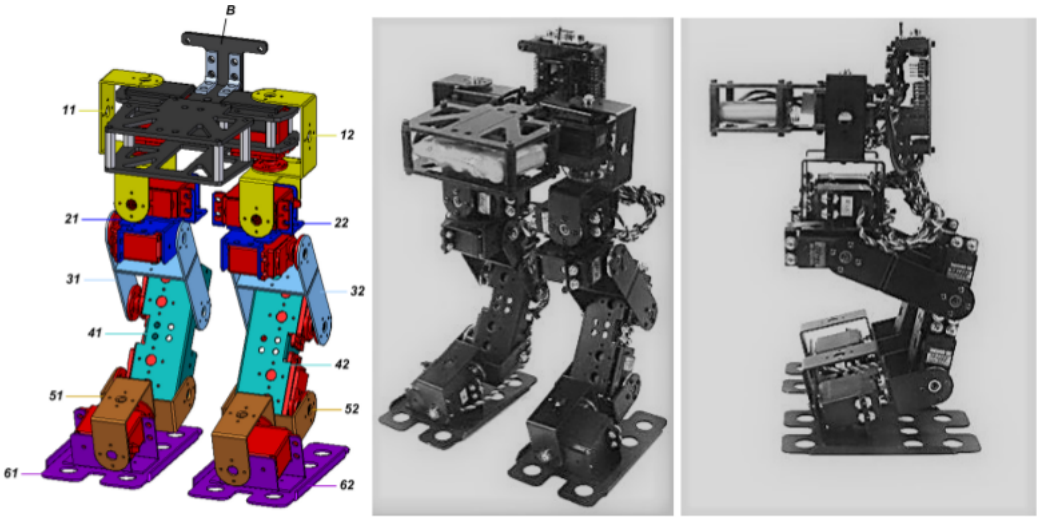

Figura 3.1: Secciones del robot bípedo Scout por Lyxmotion®[1]

4. Sensores FlexiForce®

5. Tarjeta de adquisición - Teensy 3.1®

6. Potenciómetros coaxiales con el eje de acción de los actuadores

3.3. Diagrama de conexiones

En la Figura **3.2**, se muestra el diagrama de conexiones que se utilizó para el último control de postura y caminata.

3.4. Diagrama de control

La Figura **3.3** representa el algoritmo de control de postura, el cual se pretende seguir manteniendo cuando se migre a ROS. El algoritmo es aquel flujo de procesos de obtención, cálculo y transmisión de datos. Los programas de control de postura y marcha elaboran un control PID autosintonizable.

3.5. Pruebas realizadas

Las Figuras **3.4** y **3.5** muestran las pruebas que se realizaron anteriormente junto con las variaciones arbitrarias en los ángulos Pitch y Roll (inclinación) por medio del análisis de respuestas

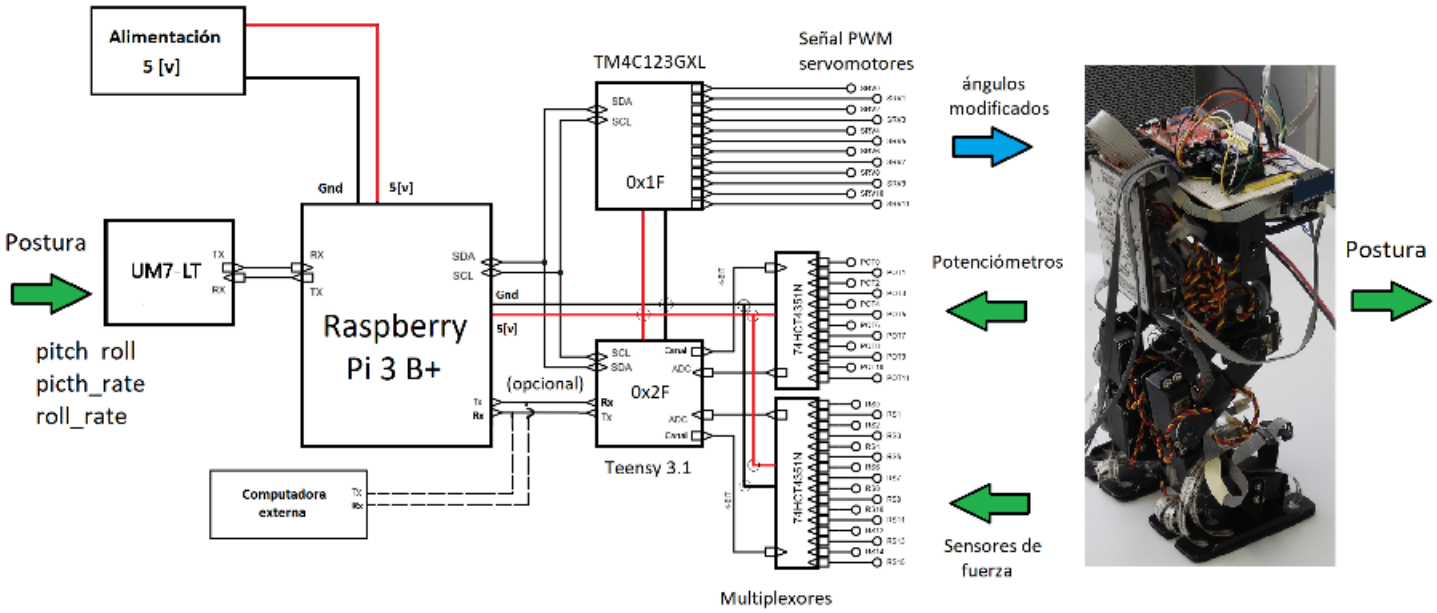

Figura 3.2: Diagrama de conexiones del sistema [1]

gráficas, dando como resultado una postura y marcha estables. Éstos proporcionan los límites de inclinación de **15°** para el control de postura y **8°** para el control de marcha [1].

3.6. Algoritmo de uso

Como se muestra en la Figura **3.2**, se tiene la transferencia y recepción combinada con RS232 e I^2C como se explica en la Sección **2.2** de protocolos de comunicación.

De dicha arquitectura *Blackboard* hubo buenos resultados, pero surgieron algunos percances ya mencionados en la Sección **1.2** del planteamiento del problema.

De manera más explícita, el algoritmo del robot bípedo del sistema anterior se basa en la Figura **3.6**.

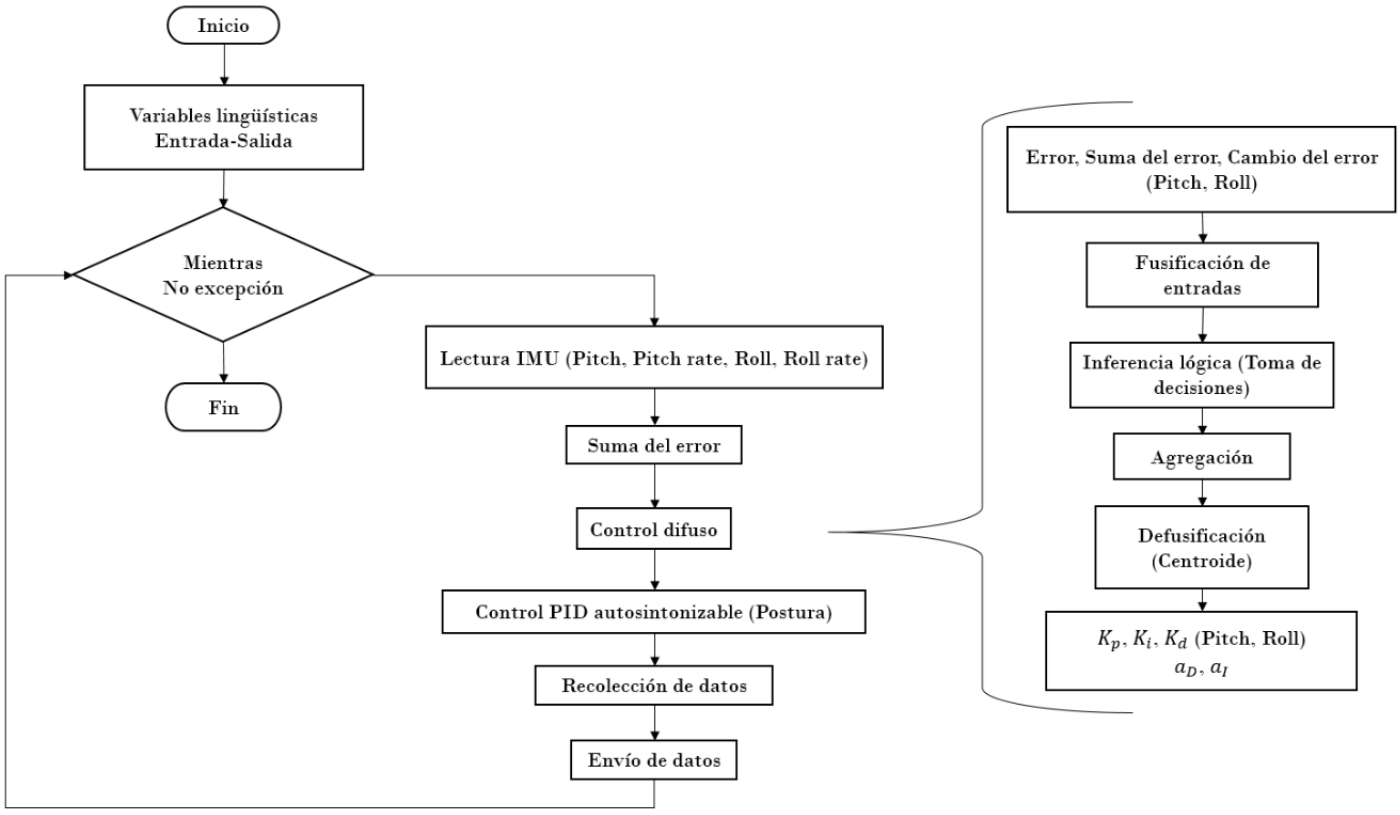

Figura 3.3: Algoritmo conceptual de control de postura [1]

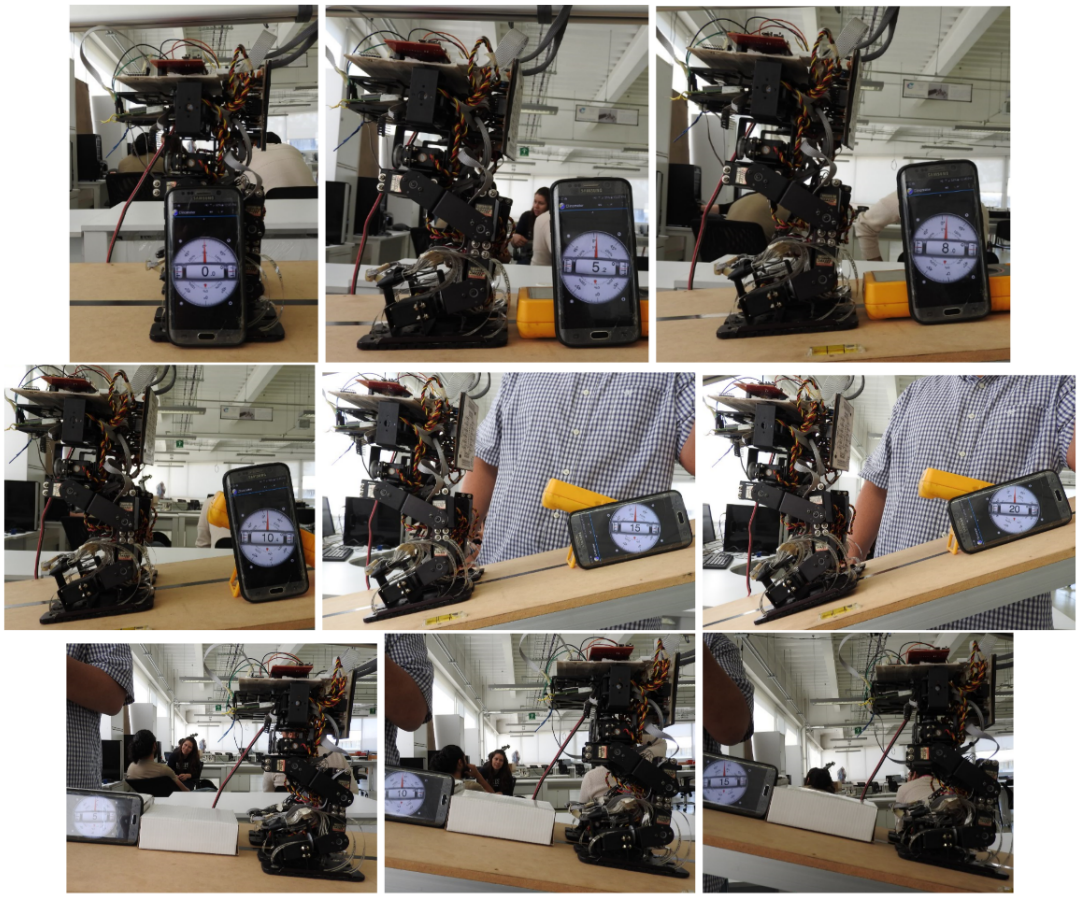

Figura 3.4: Control de postura sobre la plataforma de pruebas a diferentes grados de pendiente [1]

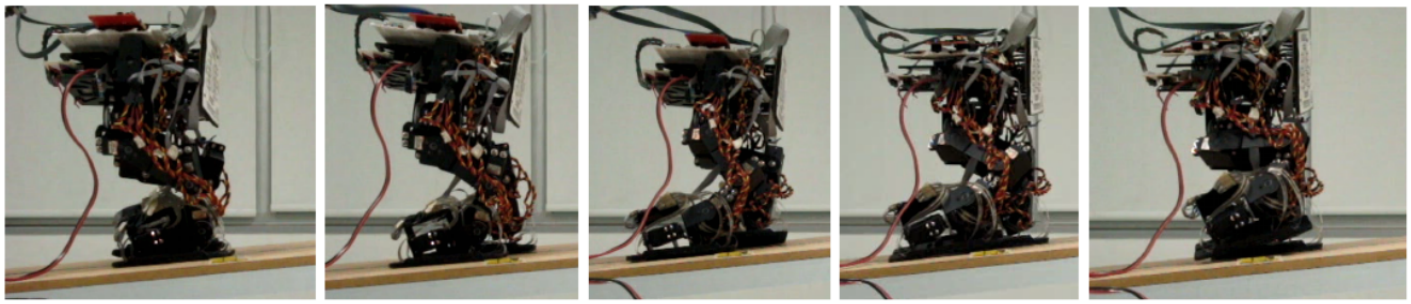

Figura 3.5: Control de marcha implementado en el robot bípedo [1]

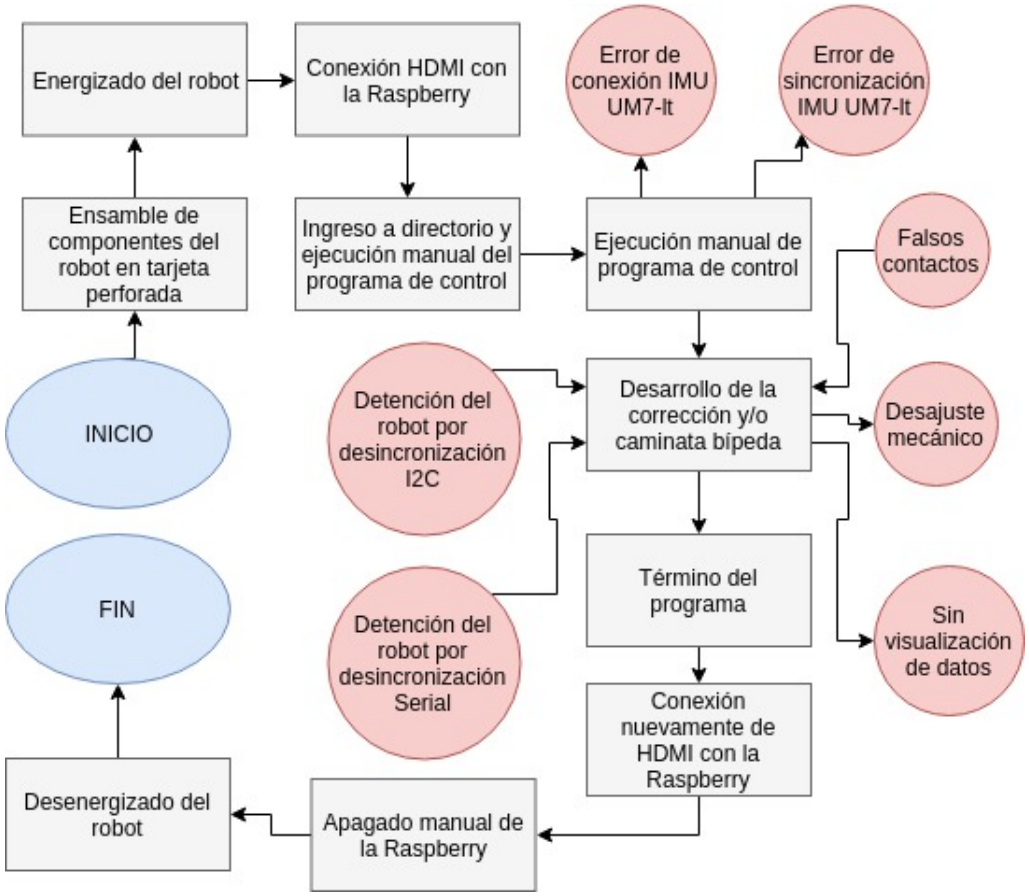

Figura 3.6: Algoritmo de uso del sistema anterior

Capítulo 4

Nueva instrumentación

En este capítulo se mencionan detalladamente los módulos necesarios para adoptar la nueva estructura con ROS ya mencionada en la Sección **2.3**. Algunos de los dispositivos se reciclaron tanto por motivos de economía como por motivos de compatibilidad.

Se propone una estrategia de solución basada en pruebas de escritorio con el propósito de disminuir las probabilidades de fallos en el sistema final, las cuales logran ser exitosas.

4.1. Raspberry Pi 3B®(CPU)

Es la Unidad Central de Procesamiento (CPU, por sus siglas en inglés) dentro de la tercera generación de las microcomputadoras lanzadas por Raspberry Pi ®(también se implementa en el esquema anterior). Algunas de las características son [1]:

- Procesador quad-core ARM Cortex-A7 de 1200 MHz
- 4 puertos USB (suficientes para esta aplicación)
- Puerto HDMI
- Ranura de tarjeta micro SD
- 1 GB en RAM

Figura 4.1: Raspberry Pi 3 Modelo B [34]

- 40 GPIO pins (Pines de propósito general de entrada/salida)
- Puerto Ethernet

4.2. UM7-lt®(IMU-Roll y Pitch)

La Unidad de Medidas Inerciales (IMU, por sus siglas en inglés) modelo UM7-lt de Pololu®también fue implementada en el sistema anterior gracias a la observación del grande error de datos en un dispositivo previo a ella. La UM7 contiene un filtro Kalman extendido (EKF) de tres ejes incorporado tanto en forma de Quaternion como de Euler. Además de ser muy estable, logra una transmisión de datos vía serial rápida y confiable para las necesidades del robot [1]. Este módulo permite dos salidas seriales UART (Rx,Tx) y un bus de comunicación SPI. También puede recibir diferentes niveles de voltaje (5V y 3.3V), donde se utiliza el puerto UART y 3.3 Volts respectivamente.

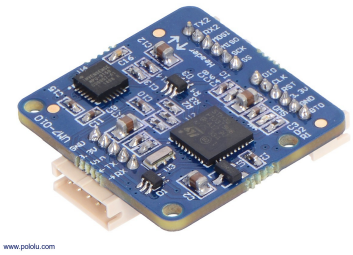

Figura 4.2: Unidad de medidas inerciales UM7-lt [33]

4.3. Tiva-C TM4C123G®(Driver de servomotores)

La tarjeta Tiva-C modelo TM4C123G es una tarjeta de desarrollo de "bajo costo"para microcontroladores basados en la arquitectura ARM Cortex M4F de Texas Instruments®. La tarjeta (que también se implementa en el esquema anterior) contiene el microcontrolador TM4C123GH6PM con una interfaz USB 2.0 y un módulo de hibernación.

Posee las siguientes características [1]:

- Microcontrolador TM4C123GH6PM de 32 bits con 80 MHz

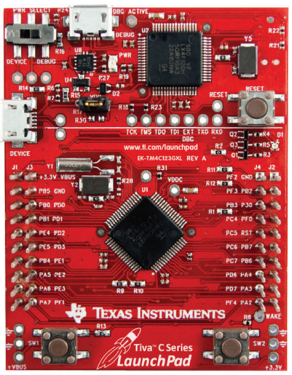

Figura 4.3: Tiva-C TM4C123G [47]

- SRAM de 32KB
- 40 pines GPIO
- Interfaz integrada de depuración
- 4 SPI
- PWMs de control de 12 bits
- Entrada USB micro
- Flash de 256KB
- EEPROM de 2KB
- Interfaz USB 2.0
- 8 UART, 2 I^2C
- 2 módulos CAN
- ADCs de 12-bits

4.4. Arduino UNO (tarjeta de adquisición)

Debido a que la tarjeta Teensy 3.1 sufrió un daño irreparable en el transcurso del desarrollo de los proyectos anteriores, se optó por adecuar la tarjeta Arduino UNO Rev3®(Figura **4.4**) al robot. La Tabla **4.1** muestra la comparación entre ambos microcontroladores.

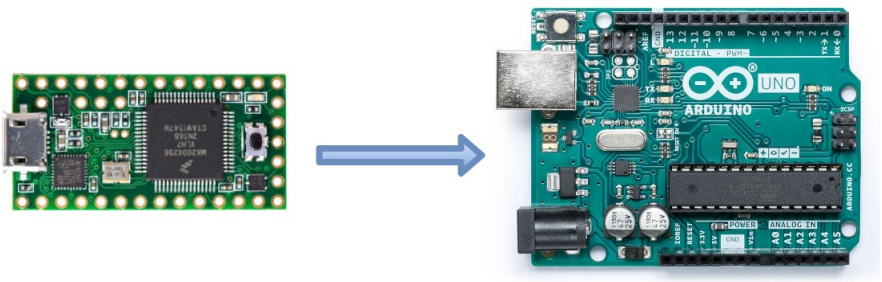

Figura 4.4: Tarjeta Teensy 3.1 [1] y tarjeta Arduino UNO R3 [43]

Queda claro que el microcontrolador de la Teensy es superior al del Arduino UNO en casi todas sus características. Sin embargo, se analizó y concluyó que las de la primera se encuentran

Tabla 4.1: Comparación entre Teensy 3.1 y Arduino UNO R3

Característica	**Teensy 3.1**	**Arduino UNO R3**
Microcontrolador	MK20DX256VLH7	Atmega328P-PU
Flash	256 KB	32 KB
Entorno de programación	Arduino IDE	Arduino IDE
Energizado	USB = 5 V o Vin = 9-12 V	USB = 5 V o Vin = 9-12 V
Reloj/Procesador	32 bit ARM Cortex-M4 72 MHz CPU	16 MHz
Voltaje en puertos digitales	5 V	5 V
UART	3 puertos Rx/Tx	1 puerto Rx/Tx
I^2C	2	1
Lenguaje de programación	C y C++	C y C++
Código abierto	Sí	Sí
Num. puertos digitales I/O	20 (6 compartidos con An)	34 (21 compartidos con An)
PWM	6	12
Precio estimado México 2018	$1000 MXN	$400 MXN

sobradas para el objetivo actual. La Teensy, a pesar de correr a 72 MHz y ser más compacta, el número de pines, los puertos seriales, e hilos, son excesivos como para optar por la compra de una nueva tarjeta. En cambio, la segunda tarjeta es más fácil de conseguir (además de ser más barata), siendo ésta la nueva encargada de la adquisición de datos de los sensores, ya que también cuenta con las características esenciales para lograrlo.

4.5. MPU-6050®(IMU-Yaw)

La IMU MPU-6050 de InvenSense®contiene tecnología de Sistemas Micro-Electro-Mecánicos (MEMs, por sus siglas en inglés) de un acelerómetro y giroscopio [35]. Es estable y precisa, además de barata en el mercado.

Contiene hardware de conversión analógico-digital de 16 bits para cada canal. Por lo tanto, captura los canales **X**, **Y** y **Z** al mismo tiempo (Roll, Pitch y Yaw). El sensor usa el bus I^2C, siendo esclavo para interactuar con el Arduino con los pines de SDA (Serial Data) y SCL (Serial Clock).

Lee los valores en bruto con un acelerómetro

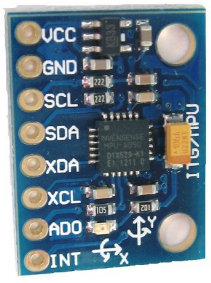

Figura 4.5: IMU MPU-6050 [35]

y un giroscopio fácilmente, pero su transformación a valores útiles no es tan sencillo. El modo de suspensión debe estar deshabilitado, y luego se pueden leer los registros del acelerómetro y del giroscopio. También contiene un FIFO buffer de 1024 bytes. Los valores del sensor se pueden programar para colocarlos en este buffer, para posteriormente leerlo con el microcontrolador. Funciona con cualquier señal de interrupción (pin 2 para el Arduino UNO).

Para el uso de este dispositivo, Jeff Rowberg del MIT (Massachusetts Institute of Technology) realizó toda una biblioteca del cálculo con filtro complementario del acelerómetro y giroscopio [35] permitiendo obtener ángulos de dirección respecto al eje Yaw como sustituto de lectura del magnetómetro de la UM7 como se explica en la Sección **2.6** de los ejes de inclinación y giro RPY y Azimuth. Modificando un poco la biblioteca de Arduino MPU-6050, esta IMU puede dar excelentes resultados para la solución de la obtención del ángulo Yaw que hace falta en la UM7-lt.

4.6. Estrategia de solución

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 propia Raspberry, se optó por realizar las pruebas preliminares en un equipo externo. Posterior a ello y con las configuraciones necesarias funcionando, todos los Nodos fueron ejecutados desde Ubuntu 16.04.04 LTS con los siguientes equipos:

- Laptop Toshiba Satellite Radius P55W con Core i5
- Laptop HP Gaming Pavilion 15-cx0001la con Core i5 7ma. generación
- Laptop HP Omen 15 con Core i7 de 8va. generación

En otras palabras, se simuló la Raspberry (Figura **4.6**) utilizando equipos con mayor probabilidad de éxito, para así enfocarse en las tareas propias de esta tesis y no en otras posibles fallas con posterior solución.
Todo lo que se encuentra en los diagramas conceptuales con una imagen de una Raspberry significa que antes de ella se simuló con una laptop.

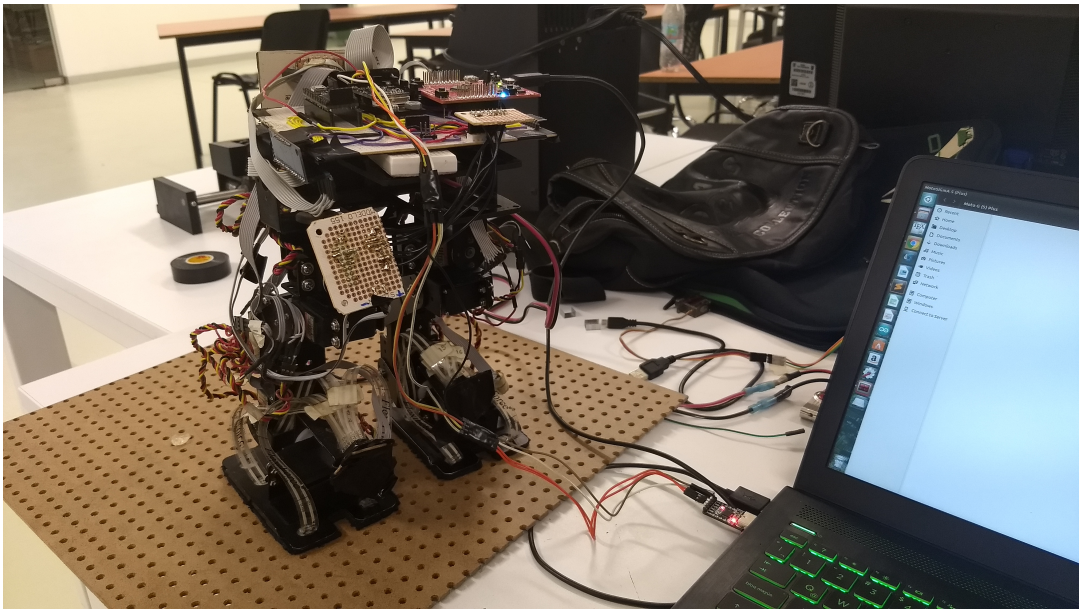

Figura 4.6: Simulación de Raspberry con Laptop HP Pavillion 15

Capítulo 5

Implementación de Nodos y Tópicos

La implementación de Nodos y Tópicos es uno de los componentes más importantes de este desarrollo de software. Para poder realizar dicha tarea es necesario categorizar y asignar las distintas variables que se transportan por medio de los Tópicos a cada uno de los Nodos que las aprovechan para cálculos, filtros, comprobaciones de datos, etc.

Finalmente, con todos los Nodos y Tópicos asignados, se realiza un diagrama conceptual completo mostrando la interacción entre la adquisición de datos y cálculos de los dispositivos, permitiendo visualizar de mejor manera la funcionalidad de la arquitectura *Peer-to-Peer* de ROS.

Asignación de color por concepto de ROS:

- Paquetes: **color verde**
- Nodos: **color amarillo**
- Tópicos: **color azul**
- Tipos de datos: **color naranja**

Para lograr la modularización de los procesos se necesitó independizar cada uno de ellos:

- Obtención de ángulos Roll y Pitch de la IMU UM7-lt, que son los que funcionan correctamente.
- Cálculo de datos en el PID autosintonizable del programa de control de postura y/o marcha bípeda desde un equipo externo (laptop).

- Escritura de ángulos de los servomotores en la Tiva-C como *esclavo*.
- Obtención de señales de los sensores Flex con el Arduino UNO (no incluidos en el control aún).
- Obtención de señales de los potenciómetros con el Arduino UNO (no incluidos en el control aún).
- Obtención del valor de ángulo Yaw de la IMU MPU-6050 con el Arduino UNO (no incluido en el control de postura ni marcha aún).

5.1. Nodo de la UM7: Roll y Pitch (inclinación)

El Paquete **um7** directamente de la ROS Wiki [48] proporciona una implementación en C++ del protocolo serial CH Robotics®con su Nodo de ROS correspondiente para publicar Tópicos estándar de inclinación y orientación llamado **um7_driver**. Éste no está integrado directamente en el módulo, sino que se convierte en Nodo una vez que recibe los datos por Rx/Tx.

Figura 5.1: Convertidor USB-TTL, para comunicación serial con la UM7 [49]

En la arquitectura *BlackBoard*, la Raspberry utilizó directamente el puerto UART con cuatro cables físicos: Rx (Recepción), Tx (Transmisión), +3.3V (Alimentación), Gnd (Tierra). Debido a que no se podía hacer uso directamente de ese recurso en una laptop sin desarmarla, se implementó un módulo USB-TTL (Figura **5.1**) y se sustituyó el correspondiente puerto por *USB0* en vez de *S0* en la instrucción de levantamiento del Nodo:
rosrun um7 um7_driver _port:=/dev/ttyS0 (*sustituido por ttyUSB0*)

Una vez ejecutada la instrucción anterior, el Paquete corrió el Nodo **um7** [48] con los siguientes Tópicos:

- **imu/data**: orientación filtrada, aceleraciones y rotaciones angulares. La orientación se especifica como Cuaternión. Sólo está disponible si se selecciona **quat_mode** (predeterminado).

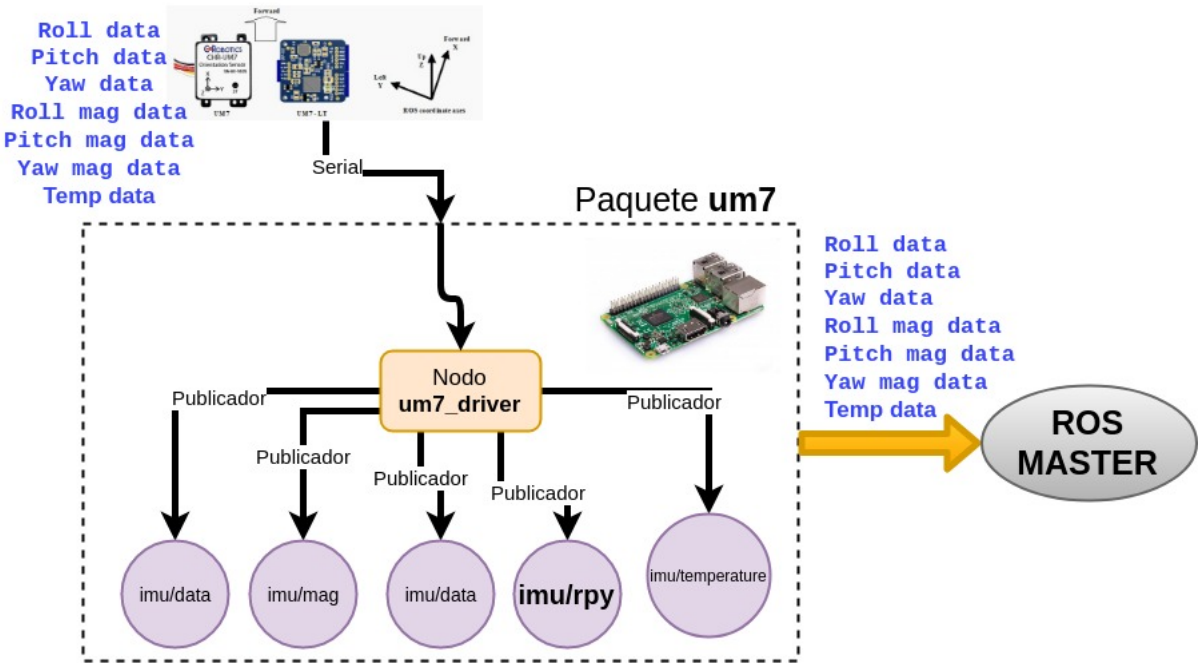

Figura 5.2: Paquete de la IMU UM7, ubicado dentro de la Raspberry

Las rotaciones angulares de los tres ejes se especifican en radianes sobre segundo. Las aceleraciones de los tres ejes se especifican en metros sobre segundo al cuadrado. Tipo de datos: **sensor_msgs/IMU**.

- **imu/mag**: datos del magnetómetro filtrados del sensor. Proporcionado como orientación 3D en **X**, **Y** y **Z**. Tipo de datos: **geometry_msgs/Vector3Stamped**.
- **imu/rpy**: angulos Roll, Pitch y Yaw de la orientación detectada. Los ángulos están especificados en radianes. Tipo de datos: **geometry_msgs/Vector3Stamped**.
- **imu/temperature**: temperatura del sensor en °C. Tipo de dato: **std_msgs/Float32**.

La velocidad de transmisión y recepción (Baudrate) de datos tanto en la IMU como en el Nodo está establecida por defecto a 115200 baudios.

El Paquete **um7** contiene al Nodo **um7_driver** que publica el Tópico de interés **imu/rpy** que se publica con una variable de tipo **Vector3Stamped**.

Finalmente, se manda con el **ROS MASTER** como se puede apreciar en la Figura **5.2**. Para ejecutarlo se escribe en la terminal:

rosrun um7 um7_driver _port:=/dev/ttyX

donde *ttyX* es el puerto correspondiente a la UM7. En la laptop es *ttyUSB0* y en la Raspberry es *ttyS0*.

5.2. Nodos de control: postura y marcha

El programa de control de postura de la tesis anterior utilizaba un bucle infinito para corregir a lo largo de la marcha bípeda las trayectorias que se generaron en el trabajo [5]. Dichas trayectorias, creadas con base en el modelo carro-mesa parametrizado, se modificaban utilizando los ángulos de inclinación de la IMU UM7 aprovechados en los controladores difusos [1].

El programa de control de marcha se enfocaba únicamente en la parte de CPID-ASLD (Control Proporcional-Integral-Derivativo-Autosintonizable mediante Lógica Difusa). En este paso se realizan los cálculos en función de la modificación de trayectorias (previamente definidas [5]) tomando en cuenta las correcciones de ángulos de inclinación como se realiza en el control de postura.

En otras palabras, el control de postura sólo corrige inclinaciones con el robot de pie y el control de marcha corrige inclinaciones con el robot caminando.

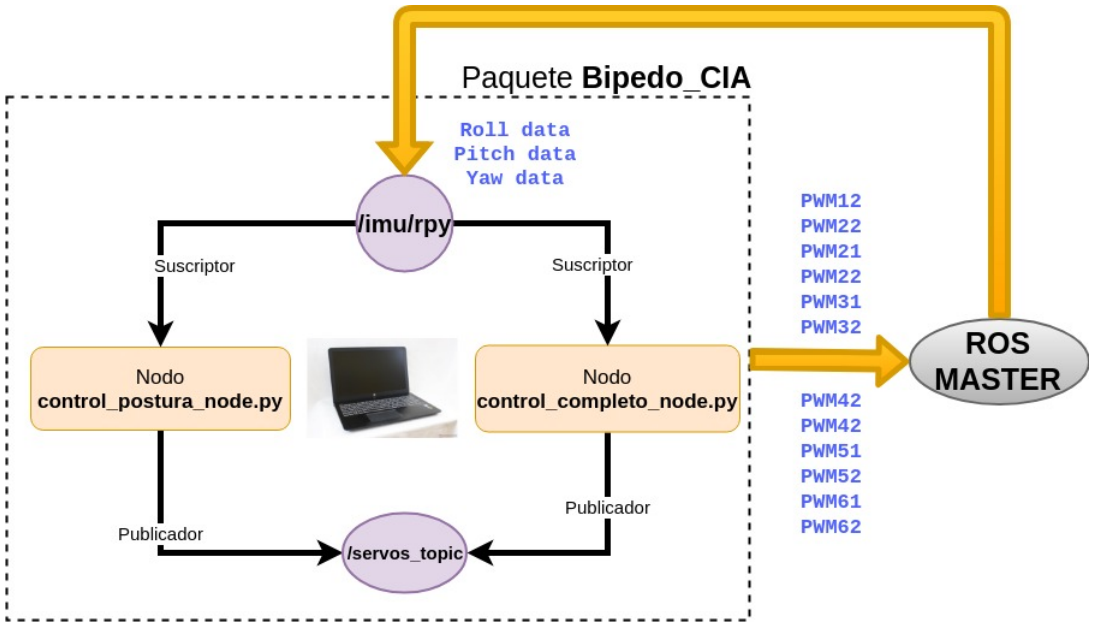

Figura 5.3: Paquete de control, ubicado en un equipo externo

Al tener los programas mencionados listos y funcionando en el sistema anterior, se adaptaron a la arquitectura de ROS. Los Nodos de Control de Postura y Control de Marcha pueden consultarse en el Apéndice **A** y **B** respectivamente. Desarrollados en Python, se suscriben al Tópico de datos de la UM7 **/imu/rpy**. A la vez, crean un publicador a un Tópico llamado **/servos_topic**, el cual publica los datos de PWM a los servomotores calculados mediante las leyes de control, los cuales se multiplican por un factor de **57.2958** para su transformación a grados.

El Paquete **Bipedo_CIA** contiene a los Nodos **control_postura_node.py** y **control_completo_node.py** quienes se suscriben al Tópico **/imu/rpy** de tipo **Vector3Stamped** y publican ángulos de **0°** a **180°** al Tópico **/servos_topic** que es de tipo **Float32MultiArray**.

Finalmente, se mandan con el **ROS MASTER** como se aprecia en la Figura **5.3**. Para ejecutarlos se escribe en la terminal:

rosrun Bipedo_CIA control_postura_ROS.py
o
rosrun Bipedo_CIA control_completo_ROS.py

5.3. Nodo de la Tiva-C: escritura de servomotores

El Nodo de la Tiva-C puede consultarse en el Apéndice **C**. Este Nodo implementado en lenguaje C para la plataforma Energia IDE [32], el cual se suscribe al Tópico **/servos_topic**, crea un publicador a un Tópico llamado **/servos_data_console**, el cual publica los mismos datos de PWM de los servomotores ya calculados en el Nodo de control con el fin de confirmar que no existe retraso notable a la hora de recibir los datos de PWM y enseguida escribirlos a los servos.

Para poder compatibilizar Energia con ROS fue necesario instalar la biblioteca **ros_lib** [31], junto con la herramienta **rosserial** [50] que sirven para la comunicación serial en Python con la Tiva-C y el Arduino UNO, ya que ambos se desarrollaron en Java.

En resumen:

- El Paquete **rosserial_python** contiene el Nodo **serial_node.py** que se inicializa a 500,000 Baudrate (especificado en el el comando de la terminal). Ayuda a comunicar a la Tiva-C con la computadora mediante conexión micro-usb con puerto *ACMX*, siendo X el puerto correspondiente del dispositivo.

- El Paquete **ros_lib** está instalado en las bibliotecas de Energia IDE pero NO contiene originalmente el Nodo **tiva_servos_node.ino** (que se programó en el código a 500,000 Baudrate). Éste se cargó directamente a la tarjeta y fue programado desde cero. El Nodo se suscribe al Tópico **/servos_topic** de tipo **Float32MultiArray** y publica al Tópico **/servos_data_console** de tipo **Float32MultiArray**.

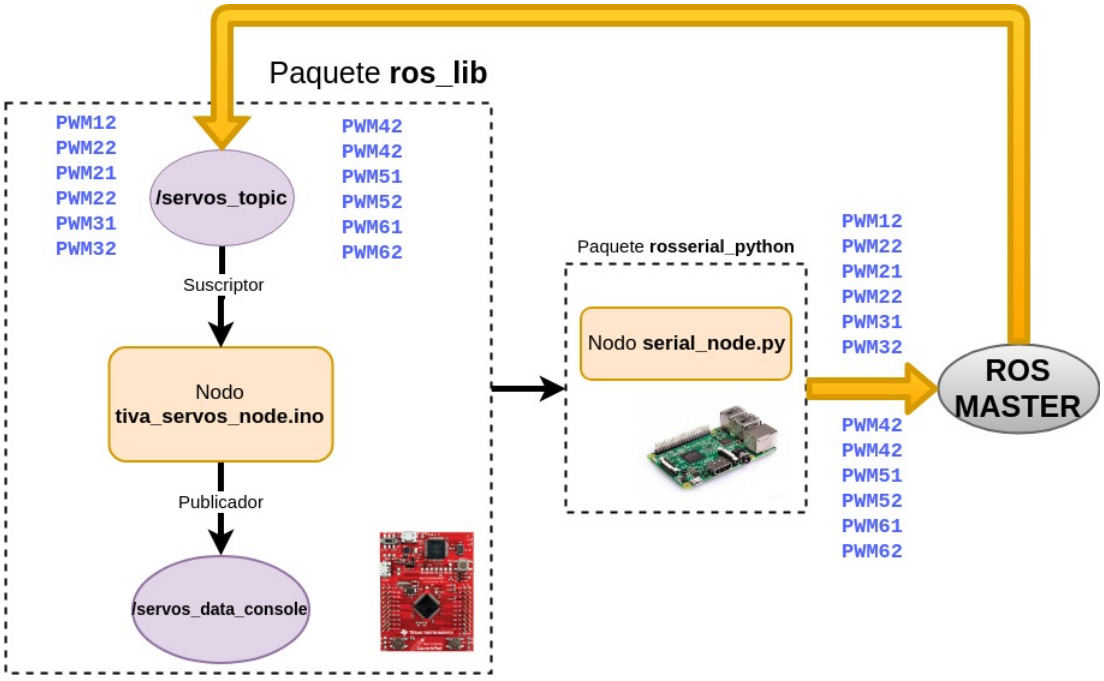

Figura 5.4: Paquete de escritura servomotores

El comando en la terminal de la Raspberry una vez conectada la Tiva-C con su respectivo Nodo con extensión .ino cargado en la tarjeta es:

rosrun rosserial_python serial_node.py /dev/ttyACM0 _baudrate:=500000

Finalmente, se manda con el **ROS MASTER** como se aprecia en la Figura 5.4.

5.4. Nodo del Arduino: lectura del eje de dirección, Flex y potenciómetros

El Nodo del Arduino se compone de dos partes: un Nodo en ROS llamado **sensores_node.py** (dentro de la Raspberry) y un programa en Arduino llamado **flex_pots_yaw_BIPEDO.ino** (cargado directamente a la tarjeta Arduino-UNO), los cuales pueden consultarse en el Apéndice D y

E respectivamente. El Nodo **sensores_node.py** implementado en Python es muy similar al de la UM7.

No se pudo aplicar el mismo principio respecto al Nodo de la Tiva, el cual es cargar directamente el Nodo en lenguaje C hacia la tarjeta, debido a que la biblioteca de Jeff Rowberg de la MPU-6050 [35] se contrapone a los recursos de ROS. No obstante, se encontró una solución inmediata al problema. Es como si se hubiese creado desde cero el Paquete **um7** pero para el Arduino, el cual recibe los datos directamente de los sensores por medio de comunicación serial.

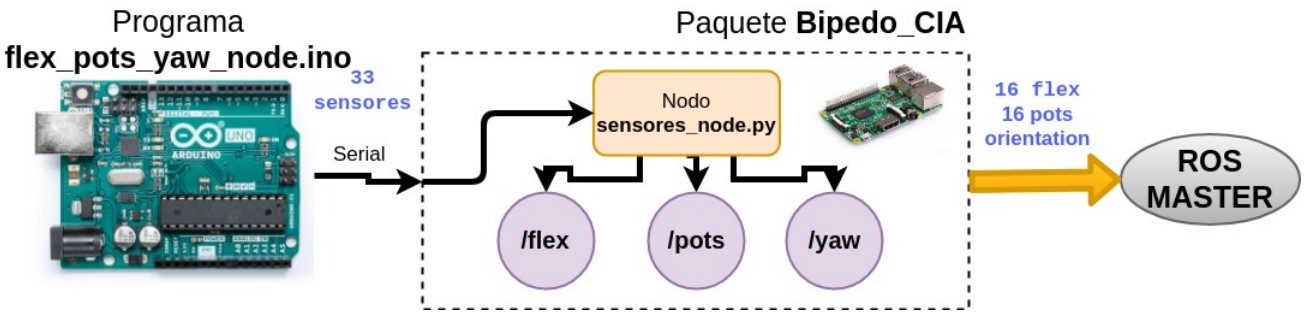

Figura 5.5: Paquete de lectura de sensores Flex, potenciómetros y Yaw

Como por el momento no es un Nodo que influya en las variables de control, **ningún Nodo tiene suscripción a sus Tópicos**, los cuales proporcionan datos de los flexómetros de las plantas de los pies, potenciómetros de los servomotores y el ángulo de dirección proporcionado por la MPU-6050.

El programa con extensión **.ino** cargado en la tarjeta concatena estos **33** valores (**16** de Flex, **16** de potenciómetros y **1** de dirección) y los manda vía serial y el Nodo **sensores_node.py** se encarga de recibir y desconcatenar la cadena, para así publicar los valores con su respectivo Tópico.

El Paquete **Bipedo_CIA** contiene el Nodo **sensores_node.py** el cuál publica a los Tópicos **/flex** de datos de sensores Flex de tipo **Float32MultiArray**, **/pots** de datos de los potenciómetros de tipo **Float32MultiArray**, y **/yaw** de dato de dirección de tipo **Float32** los datos recibidos del programa **flex_pots_yaw_node.ino**.

Finalmente, se manda con el **ROS MASTER** como se aprecia en la Figura **5.5**.

Nota importante: La calibración de la MPU-6050 requiere un tiempo aproximado de 20 segundos sin mover al robot para que tome su referencia como 90° (viendo hacia enfrente).

5.5. Configuración del ROS-MASTER

Existe una forma muy cómoda de trabajar remotamente con un equipo externo (equipo de escritorio o laptop), visualizando los Tópicos y Nodos sin necesidad de entrar directamente a la CPU principal que contiene el ROS MASTER (Sección **2.5.5**). Para lograr esto hay que estar necesariamente conectado a una red local (con o sin internet) en ambos equipos, ya que ROS trabaja con sockets.

La Figura **5.6** describe mejor la configuración del ROS MASTER. Como se puede observar, se encuentra alojado en la Raspberry, pero gracias a la arquitectura *Peer-to-Peer*, se ignora la distinción entre ambos, permitiendo que cualquier Nodo deseado pueda interactuar con cualquier Tópico de interés.

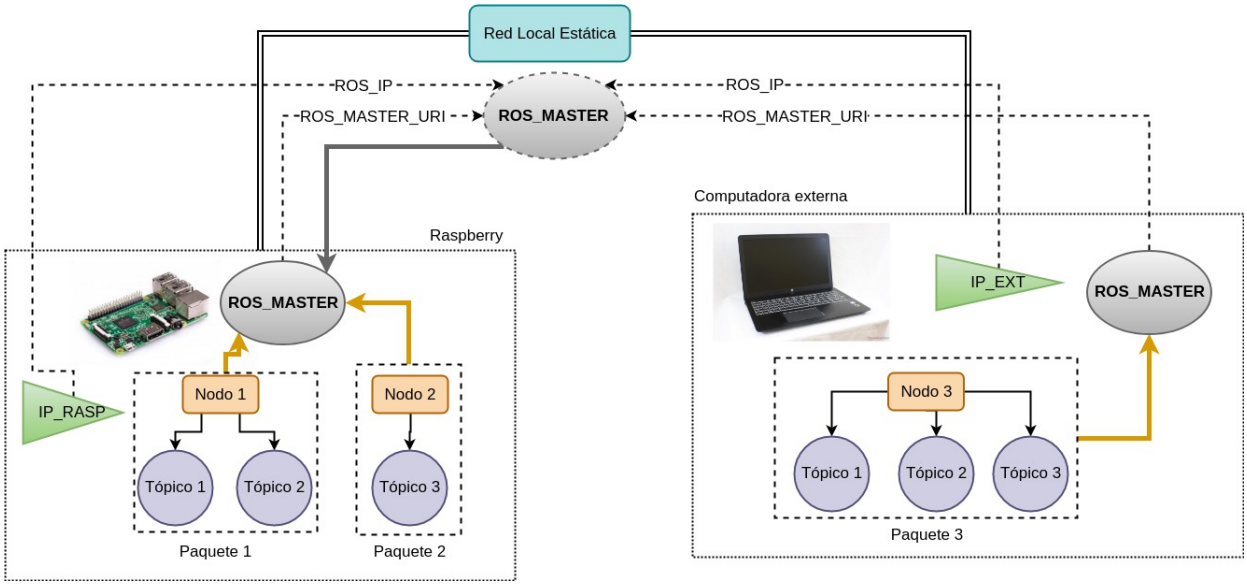

Figura 5.6: Diagrama conceptual de configuración de ROS_MASTER y ROS_IP para establecer monitoreo remoto

Se decidió que la Raspberry tuviera el ROS MASTER debido a que se puede tener una IP estática con un router, permitiendo que los usuarios no tengan la necesidad de buscarla en cada ocasión que se utiliza el robot. Además, de esta forma se reduce la serie de comandos, ya que al ejecutar un archivo **.launch**, el comando **roscore** se ejecuta automáticamente, situación que no ocurre cuando es de forma manual teniendo el ROS MASTER en una computadora externa.

Esto es más entendible conceptualmente para los futuros usuarios, sabiendo que se usa la Raspberry como módulo *esclavo*, porque obtiene únicamente datos de los sensores y los publica en sus

respectivos Tópicos, al igual de que se suscribe al Tópico de datos motrices de los servomotores. Esto permite a las computadoras externas trabajar únicamente en las leyes de control y cálculo de trayectorias, generando una mejor administración sobre los programas, modularizando procesos y dispositivos. En este punto, ya se pueden realizar trabajos colaborativos entre varias personas.

Para poder configurar el ROS MASTER se necesitaron establecer 4 parámetros en la terminal: dos en la Raspberry y dos en la computadora externa. Recordar que para ver la IP del equipo se utiliza el comando **ifconfig**:

En la Raspberry

Asignar el ROS_MASTER con la IP de la Raspberry (la cual contiene al **roscore**):
export ROS_MASTER_URI=http://*IP_RASPBERRY*:11311
Asignar el ROS_IP con la IP de la Raspberry:
export ROS_IP=*IP_RASPBERRY*
Por ejemplo, considerando la IP estática de la Raspberry obtenida con el router del proyecto:
export ROS_MASTER_URI=http://172.16.9.147:11311
export ROS_IP=172.16.9.147

En el equipo externo

Asignar el ROS_MASTER con la IP de la Raspberry (que tiene al **roscore**):
export ROS_MASTER_URI=http://*IP_RASPBERRY*:11311
Asignar el ROS_IP con la IP de la computadora externa:
export ROS_IP=*IP_COMPUTADORA_EXTERNA*
Por ejemplo, considerando la IP estática de la Raspberry obtenida con el router del proyecto y la IP autoasignada a la HP Pavilion Power Laptop:
export ROS_MASTER_URI=http://172.16.9.238:11311
export ROS_IP=172.16.9.169

Es muy importante asegurarse de que todos los parámetros se encuentren en orden, ya que si alguno falla, simplemente no se podrán visualizar los Tópicos y Nodos, aún funcionando el robot. Sin embargo, esto no es de preocuparse, ya que se contará con una computadora de escritorio con todo el respaldo incluyendo los parámetros de conexión (ROS_MASTER y ROS_IP) ya configurados para evitar fallos. **Lo anterior mencionado es sólo por si se requiere conectar con ROS al robot y una laptop** con los archivos .bashrc de la laptop (Apéndice **F**) y .bashrc de la Raspberry (Apéndice **G**).

5.6. Diagrama y algoritmo actual de uso

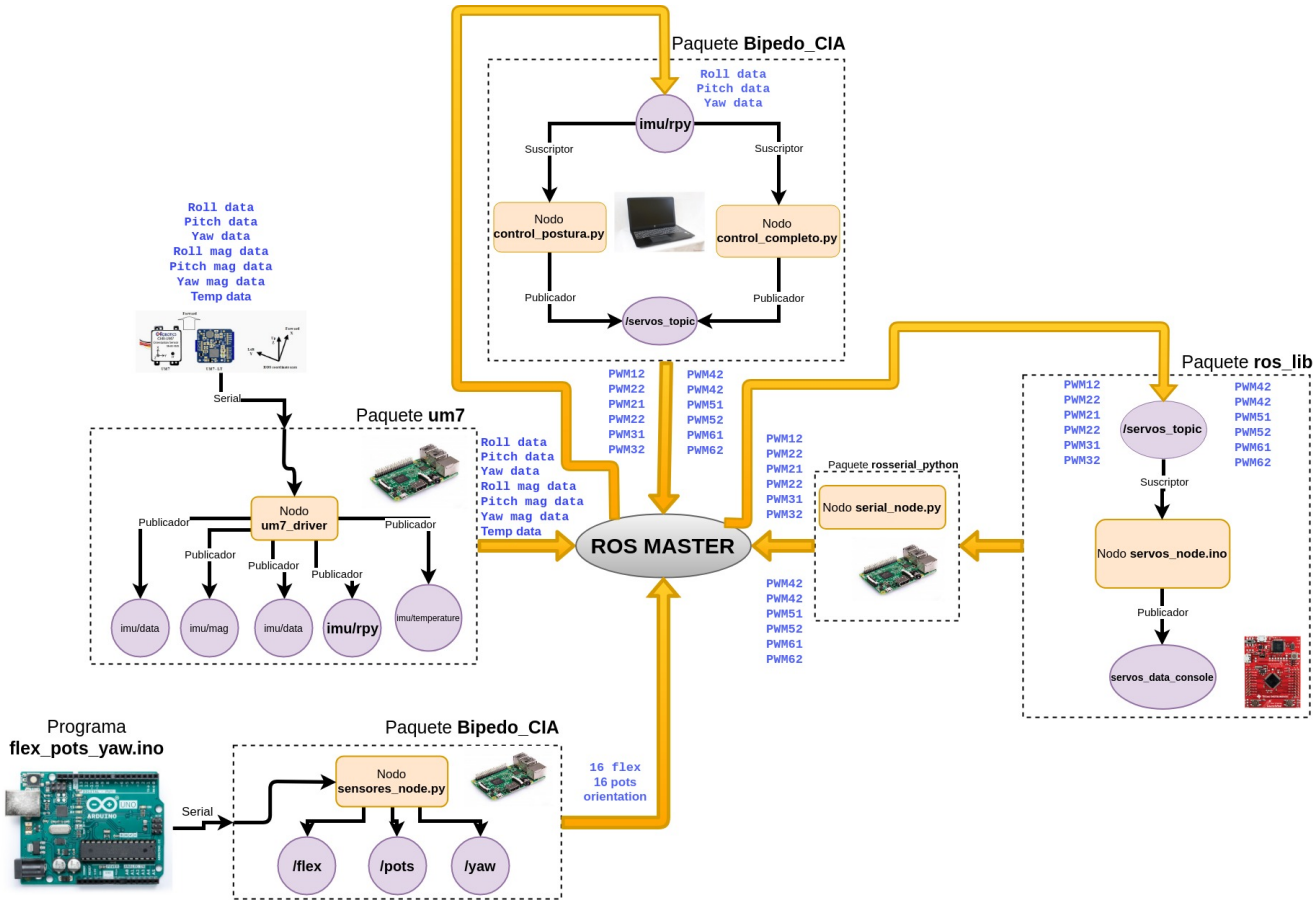

Figura 5.7: Diagrama conceptual completo de Nodos y Tópicos del Bípedo

El diagrama conceptual completo del conjunto de Nodos y Tópicos se muestra en la Figura **5.7**.

ROS tiene el objetivo de simplificar mucho trabajo y llevar a cabo un control TOTAL sobre todos los procesos del robot (entre otras ventajas), logrando un adecuado diagnóstico y solución a cada uno de los problemas presentados en el planteamiento del problema.

La Figura **5.8** presenta el algoritmo final de uso que tendrá el robot de ahora en adelante.

Comparándolo con el antiguo diagrama de la Figura **3.6**, se puede apreciar claramente que la propuesta presentada, además de reducir el número de pasos a seguir para operar el robot, permite hacer uso de muchas más herramientas para futuros desarrolladores.

Cabe destacar que si algo falla en ambos algoritmos (especialmente variables de control), el robot presentará un malfuncionamiento. Sin embargo, la gran diferencia entre ellos es que los procesos (Nodos y Tópicos) seguirán siendo visuales en el nuevo sistema debido a su independización

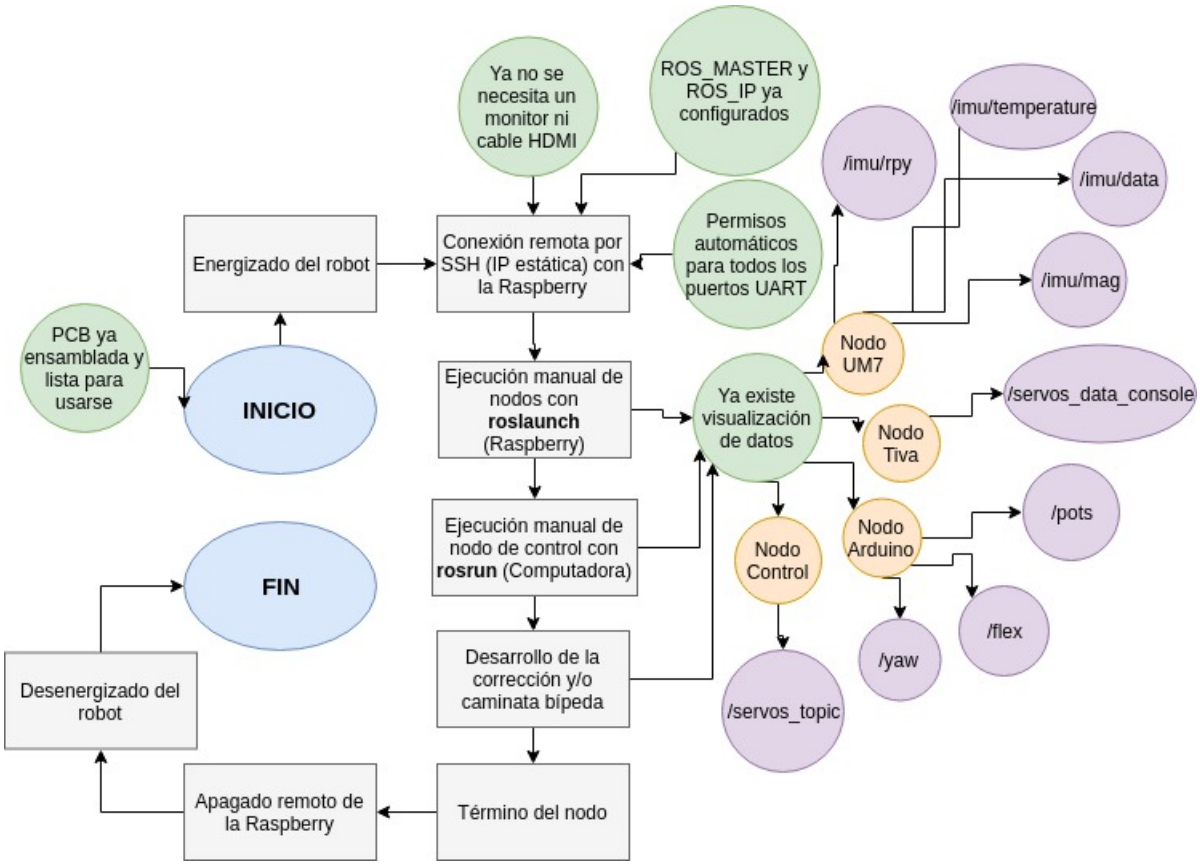

Figura 5.8: Nuevo algoritmo de uso con Tópicos y Nodos

modular e integración de programas. En cambio, en el sistema anterior se imposibilita identificar a tiempo y con precisión la falla presentada.

Capítulo 6

Diseño y manufactura de PCB

Teniendo claro el diagrama conceptual y el funcionamiento de cada Paquete de ROS, se procedió a realizar el diseño para la manufactura de la nueva Tarjeta de Circuito Impreso (PCB, por sus siglas en inglés) en el software Eagle [37]. Esta tarjeta comunica todos los elementos del robot por medio de los protocolos de ROS con cables físicos y conexiones con pistas de cobre.

6.1. Diseño

Como analogía del diagrama de conexiones del sistema anterior (Sección **3.3**), se procedió a diseñar uno con los nuevos requerimientos del sistema.

Se realizaron dos diagramas; el primero de la Figura **6.1** es como tal el nuevo diagrama de conexiones creado en el software de diseño electrónico. El segundo de la Figura **6.2** es una representación gráfica del primero.

Posterior a ello, llegó la etapa del acomodo de los componentes y diseño de pistas. A este archivo se le llama Board (Tarjeta), ya que es la representación física de la nueva PCB al final de la manufactura como se muestra en la Figura **6.3**.

Algunas de las consideraciones que se tomaron en cuenta para llegar a la propuesta final fueron:

- La MPU-6050, al ser una IMU, necesitaba estar lo más cerca del centroide del robot desde una vista superior [1], así que se decidió colocarla justo debajo de la UM7.

- Los dos microcontroladores, el Arduino y la Tiva, se colocaron lo más simétricos posible dentro de la tarjeta. Esto con el propósito de facilitar al usuario el ensamble de los módulos y disminuir el peso del robot, afectando lo menos posible la configuración electrónica anterior.

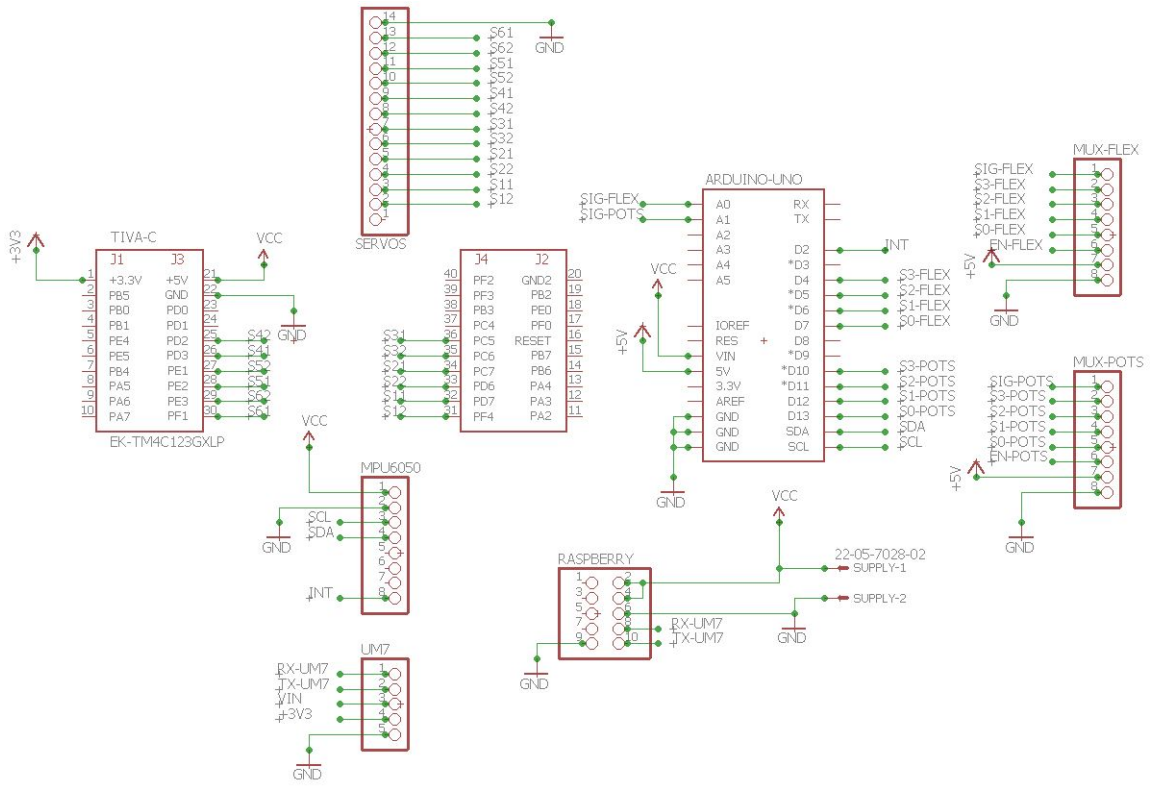

Figura 6.1: Diagrama esquemático de conexiones electrónicas en EAGLE

- Los headers de los multiplexores y de los potenciómetros se quedaron prácticamente en la misma posición, ya que, además de facilitar las conexiones internas en la PCB, ayudaron al ensamble de los tres módulos (Tiva-C, Arduino y Raspberry).

- Se creó un bus de datos de diez pines con un paquete de 5x2 (con su respectiva alimentación y tierra) hacia la Raspberry. Así, existe menos probabilidad de desconexiones y falsos contactos, ya que los demás módulos pueden desconectarse a voluntad, pero si no se realiza el proceso de apagado correcto de la Raspberry, puede llegar a corromperse la imagen o simplemente dañarse (ocurrió una vez en los primeros meses de pruebas).

- Se procuró que la nueva PCB fuera lo más similar posible a la anterior (perforada) únicamente reduciendo su tamaño y colocando correctamente los agujeros de ensamble. Esto con objeto de reducción de costos.

- Se diseñó de forma medianamente modular. Es decir, dejando puros conectores macho y hembra para los dispositivos y alimentación del sistema, pero sin disponer de todos los pines de los tres dispositivos.

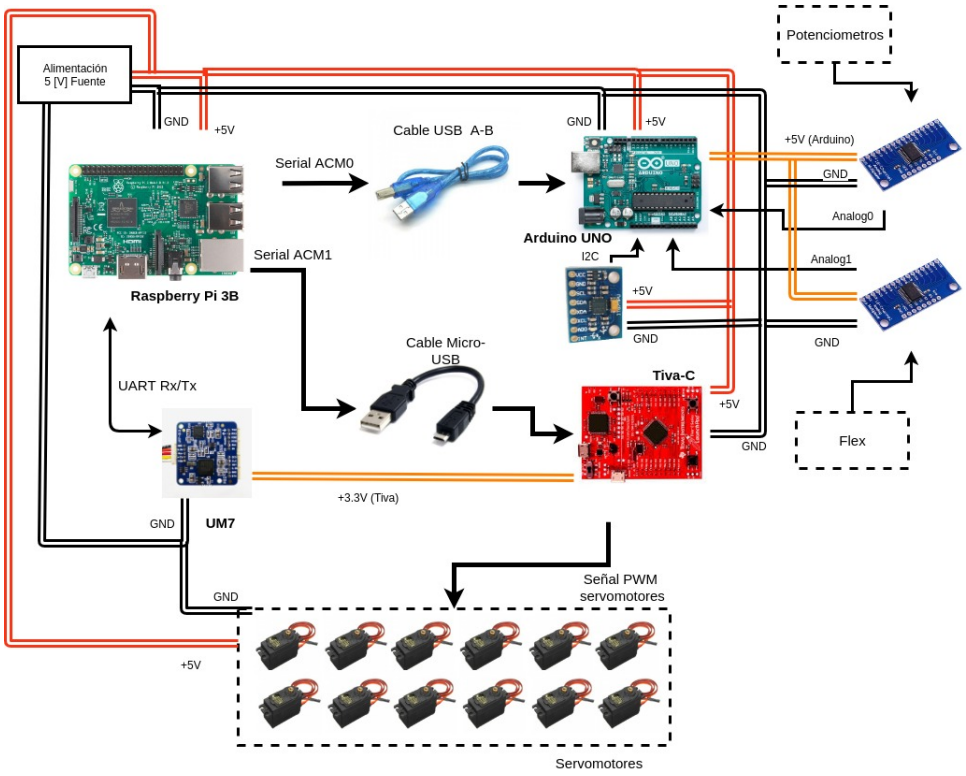

Figura 6.2: Representación gráfica a mayor detalle del diagrama esquemático

6.2. Manufactura

El prototipo de la Figura **6.4** y la Figura **6.5** se mandó a fabricar con un router de Control Numérico Computarizado (CNC) sobre un laminado de cobre de doble cara FR-4 (fibra de vidrio), un material de calidad y durabilidad excelentes.

6.3. Soldadura y acabado

Una vez comprados todos los conectores, incluyendo los buses de datos y el cable plano, se procedió a soldar y barnizar la PCB como se aprecia en las Figuras **6.6** y **6.7**. Esto con el fin de aislar los pads y pistas de posibles cortos circuitos, ya que dan lugar a las uniones entre conectores, cables físicos y el circuito impreso. Se considera que se tomaron buenas decisiones, ya que el nuevo diseño tenía que superar la tarjeta perforada anterior sin llegar al costo máximo que vendría siendo con metalizado de vías y máscara antisoldante (mayor estética y seguridad). Este prototipo con todo y componentes tuvo un costo aproximado de $300 MXN, mientras que a nivel industrial con todo lo previamente mencionado varía entre $800 y $4000 MXN.

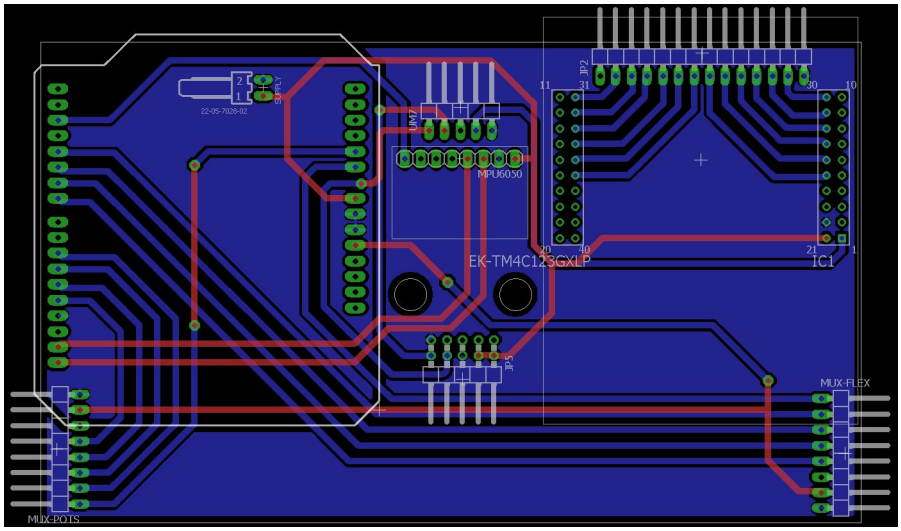

Figura 6.3: Diseño físico de la nueva PCB

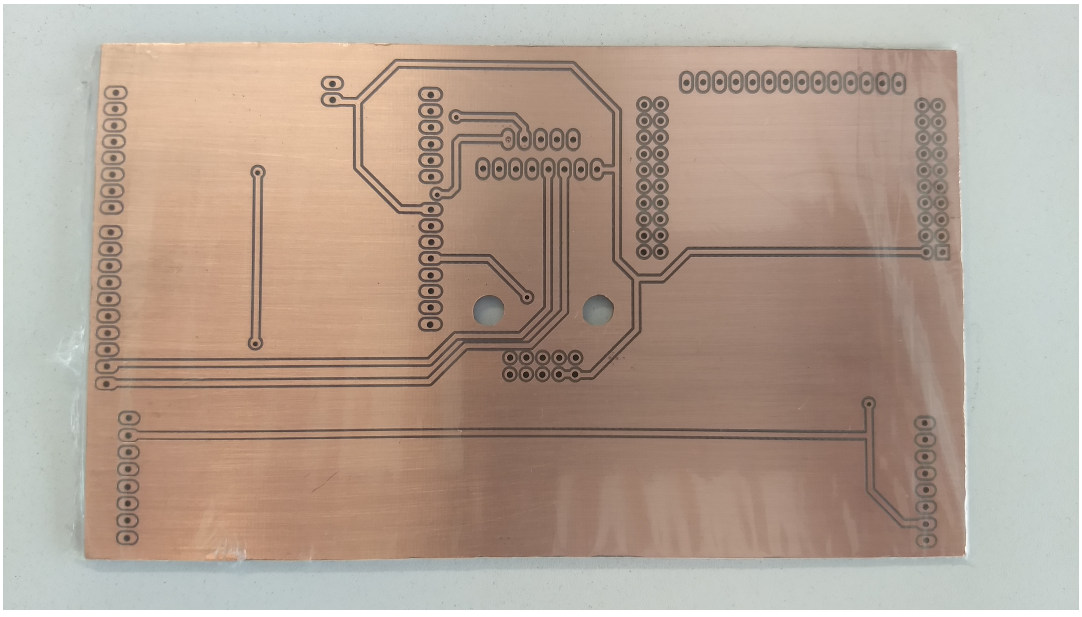

Figura 6.4: Manufactura del circuito impreso, parte top

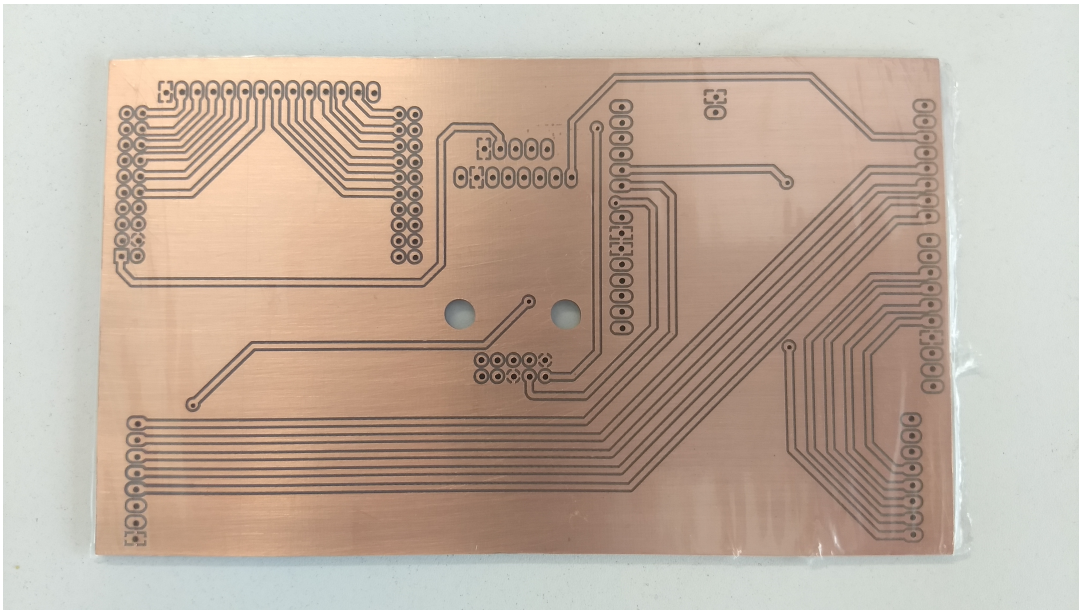

Figura 6.5: Manufactura del circuito impreso, parte bottom

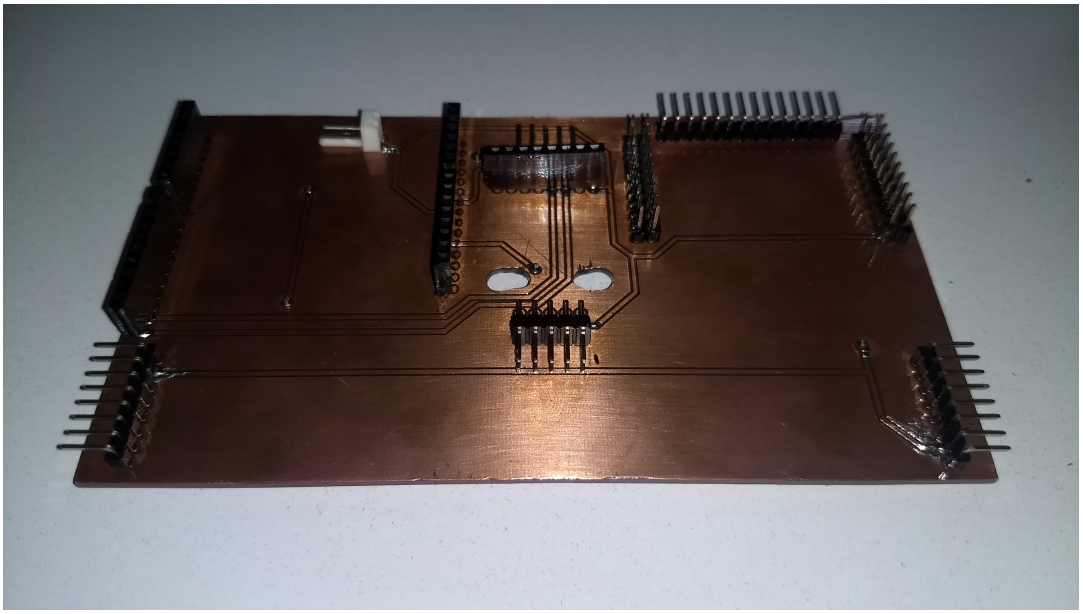

Figura 6.6: Soldado y acabado del circuito impreso, parte top

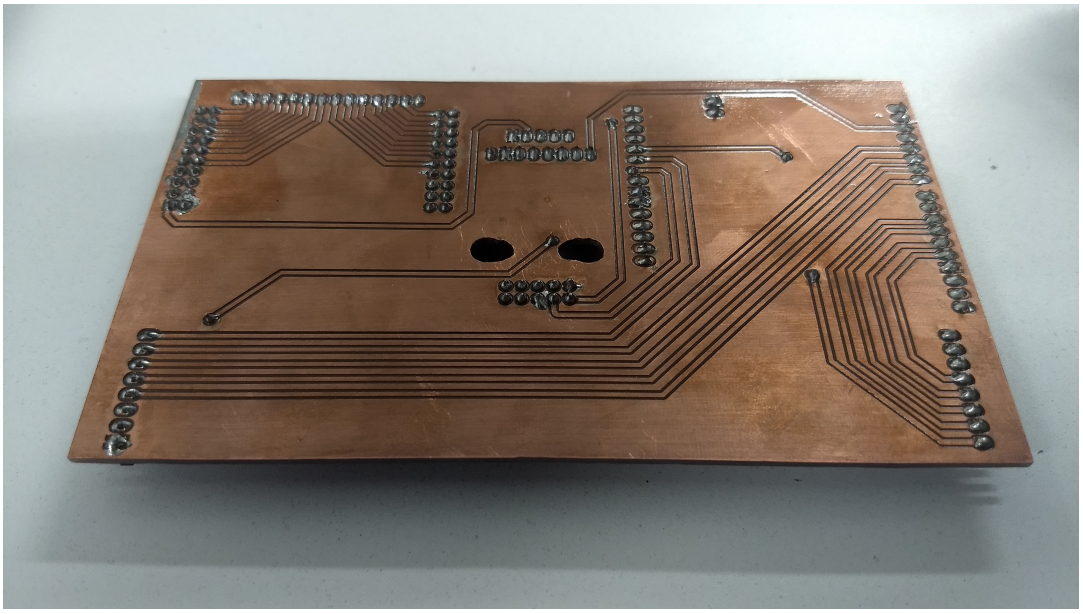

Figura 6.7: Soldado y acabado del circuito impreso, parte bottom

Capítulo 7

Preparativos finales

En el presente capítulo se pulen los detalles finales para tener toda la plataforma virtual lista para emplearse, junto con una pequeña guía de pasos de ejecución para la operatividad del robot bípedo.

7.1. Instalación de ROS Kinetic y sus herramientas

Una vez simulada la Raspberry con las pruebas de escritorio mencionadas en la Sección **4.6** con un equipo externo, de procedió montar una imagen de Ubuntu MATE [39] en una memoria SD de 16 GB en la Raspberry. En ésta se instaló ROS **Kinetic Frame** [29] y la herramienta **rosserial** [31] para comunicar tanto la Tiva-C como el Arduino-UNO con la Raspberry.

7.2. Identificador único de uso de puertos

Frecuentemente, el sistema presente confusiones cuando ejecuta los Nodos en ROS ya que los dispositivos no suelen tener un identificador único cuando se conectan al CPU. Por ejemplo, existe confusión al levantar el Nodo del Arduino puesto que el robot *intuye* que se identifica como el de la Tiva, o viceversa, debido a que existe confusión entre el puerto *ACM0* y el *ACM1*. Esto no es funcional ya que ambos tendrían que estar configurados a la misma transmisión de datos (Baudrate) de 500000 o 115200. Además, el Arduino no cuenta con un Nodo integrado como bien se explicó en el desarrollo de Nodos de la Sección **5.4**, a diferencia de la Tiva, que sí cuenta con ello.

Es necesario buscar un dato que indique alguna propiedad única del dispositivo, para que no importe cuantas veces se conecte, siempre se debe reconocer el mismo y asignarle un lugar específico

en la lista de puertos.

La herramienta **udevadm** da la solución precisamente para lo que se busca. Para ello se tiene que editar un archivo de extensión **.rule** que viene integrado en el sistema [38]. El siguiente comando sirve para ver la lista de atributos del dispositivo (se probó con el Arduino):
udevadm info –name=/dev/ttyACM0 –attribute-walk
Buscando algún atributo único, se encontró *ATTRSidVendor==‘2341’*.
Posterior a ello, se modifica el siguiente archivo con el editor de texto **nano**:
sudo nano /etc/udev/rules.d/99-usb-serial.rules
Y se agrega la línea al final del documento utilizando el identificador:
KERNEL==“ttyACM”,ATTRS{idVendor}==“2341”,SYMLINK+=“Arduino-UNO”*
y para la Tiva-C
KERNEL==“ttyACM”,ATTRS{idVendor}==“1cbe”,SYMLINK+=“Tiva-C”*
Hecho eso, se guarda y se cierra el archivo, y es necesario cargar nuevamente el archivo **.rules** en una nueva terminal:
sudo udevadm control –reload-rules
Al principio se pueden llegar a tener pequeños problemas, pero para realizar esto se recomienda, una vez ejecutada la última línea, desconectar y conectar el dispositivo para que logre reconocerlo con:
ls /dev (En este caso se llamaron *Arduino-UNO* y *Tiva-C*).

La actual imagen de la Raspberry cuenta con sus propios identificadores. Si en el futuro se pretende cambiar de dispositivo, hay que recordar que el identificador único es eso, *único*, por lo que habrá que encontrar alguna propiedad exclusiva del nuevo dispositivo a conectar.

7.3. Uso del puerto UART Rx/Tx de la Raspberry

Para lograr que la Raspberry reconozca a la UM7 como el dispositivo serial oficial directo a los pines UART Rx/Tx (puerto *ttyS0*) para eliminar el convertidor USB-TTL de la Figura **5.1**, se requiere editar el archivo */boot/config.txt* cambiando la línea *enable_uart=0* a *enable_uart=1* y asignando la frecuencia a *core_freq=250*. Finalmente, se tienen que dar permisos de escritura al archivo:
sudo chmod +x /boot/config.txt

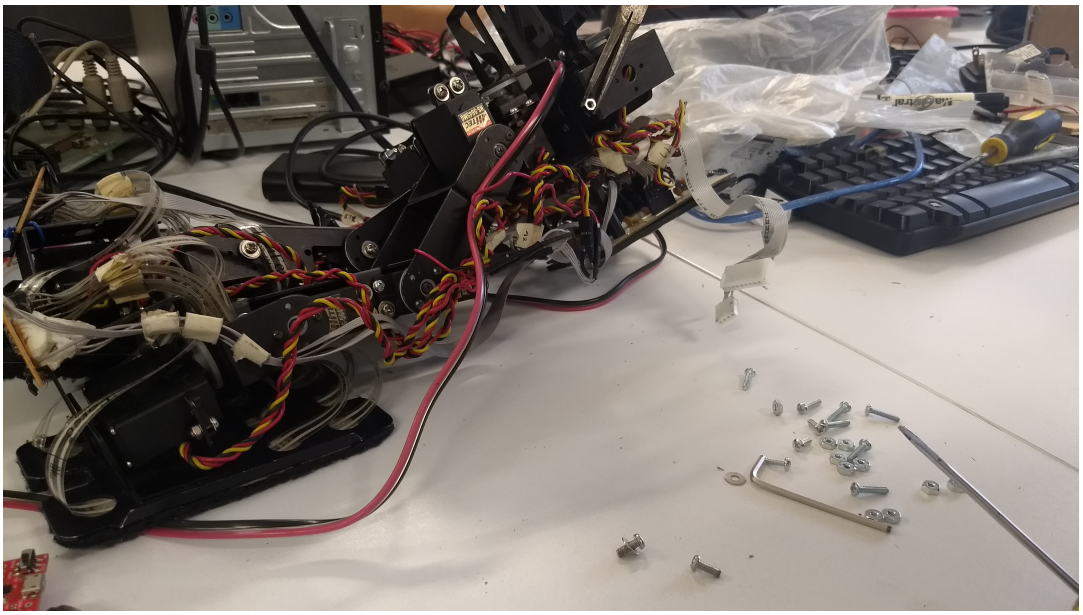

Figura 7.1: Mantenimiento mecánico con reensamble de tornillos, eslabones y actuadores

7.4. Mantenimiento mecánico

El desajuste de tornillos es uno de los grandes causantes de fallas en la marcha bípeda porque genera un juego mecánico entre eslabones, permitiendo movimientos erróneos y perturbaciones en los datos de inclinación por cada pisada. No sólo transfiere información inútil, sino que también causa que el robot caiga al suelo y aumente la tasa de desconexiones electrónicas.

Se observó que varias uniones se encontraban en estado fatal. Además, es complicado acceder a ellas para ajustarlas de manera rápida, por lo que se decidió desarmar algunos eslabones del robot y dejarlo lo más asegurado posible para las pruebas del proyecto como se muestra en la Figura **7.1**.

7.5. Corrección de postura inicial

Con el paso de los años, se generó un desgaste interno considerable en los actuadores como producto de la fatiga electromecánica. Esto causó un desajuste en los ángulos de postura inicial del robot, los cuales afectaron directamente al control de marcha bípeda reduciendo su eficacia.

Para evitar falsas trayectorias se propuso ajustar nuevamente la postura inicial del robot, intentando que la UM7 se encontrara lo más horizontal posible como en la Figura **7.2**.

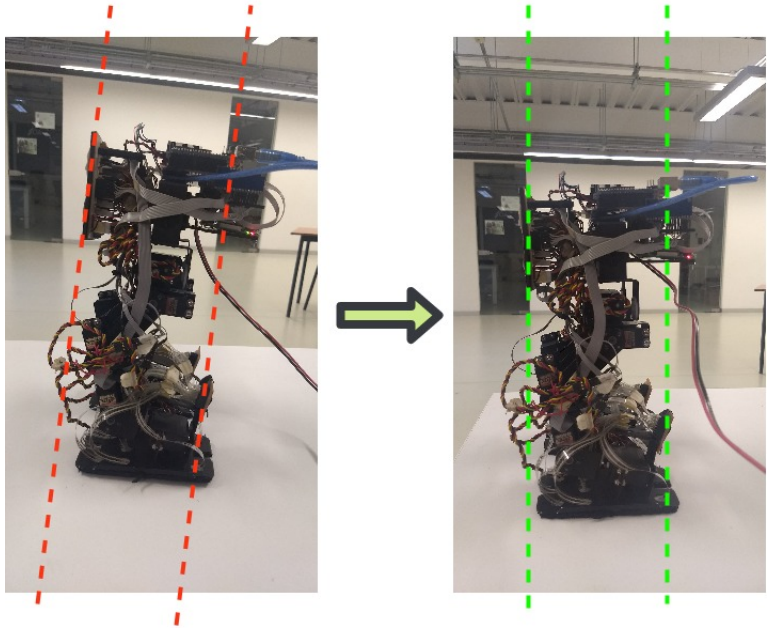

Figura 7.2: Corrección manual de valores iniciales de postura

7.6. Roslaunch del Bípedo

Los archivos **.launch** ayudan a iniciar múltiples Nodos junto con el ROS MASTER, así como sus especificaciones de transmisión de datos, puertos, etc.

Programado en formato **XML**, el archivo llamado *bipedo.launch* se encuentra en el Apéndice H. Éste se encarga de levantar:

- El Nodo **um7_driver** del Paquete **um7** con el puerto **ttyS0** (UART Tx/Rx ya configurado en la sección anterior) a velocidad de transmisión de **115200** por defecto de la IMU
- El Nodo **sensores_node** del Paquete **Bipedo** con el puerto **Arduino-UNO** a velocidad de transmisión de **115200** programado en el código
- El Nodo **serial_node** del Paquete **rosserial_python** con el puerto **Tiva-C** a velocidad de transmisión de **500000** programado en el código

7.7. Shell Script de inicio automático del Bípedo

Un Shell Script es un archivo de texto con extensión **.sh** que toma la información escrita como líneas de comandos en la terminal [41].

Los pasos para escribir un Shell Script son los siguientes:

- Escribir el archivo **.sh** con el editor de texto preferido
- Guardarlo en alguna ubicación conocida, recomendando ser ésta */Home* o alguna de sus subcarpetas para evitar confusiones
- Darle permisos al archivo con **sudo chmod 777***/UBICACIÓN_DEL_ARCHIVO/NOMBRE.sh*

En la estructura básica de un Script se necesita utilizar como primera línea:
#!/bin/bash

Este comando es reconocido como *shebang*, el cuál le indica al script qué programa es utilizado para interpretar el archivo, el cual en este caso es **/bin/bash**. Algunos otros lenguajes de programación utilizan este mecanismo, como lo son Tk y Python.

Posterior a esa línea, sólo resta ejecutar el script. Linux tomará todo lo escrito en ese archivo como si manualmente se ingresaran los comandos en una o varias terminales.

El robot requiere de la ejecución de varios comandos en la terminal para levantar todo lo que ROS necesita:

- Permisos de escritura/lectura para el puerto del Arduino UNO (Sección **7.2**)
- Permisos de escritura/lectura para el puerto de la Tiva (Sección **7.2**)
- Permisos de escritura/lectura para el puerto de la UM7 (ttyS0)
- Configurar el ROS MASTER dentro en una IP estática en una red local para poder visualizar los procesos de manera remota (Sección **5.5**)
- Agregar las variables de entorno que ROS necesita para funcionar por medio del comando **source /setup/devel.bash**

Actualmente todos estos requisitos se encuentran agregados al script llamado */home/Scripts/-Bipedo.sh*, que puede consultarse en el Apéndice **I**. Podría decirse que el script tiene una función parecida a la de los archivos **.launch**, que ejecutan varias instrucciones con sólo escribir un comando. Sin embargo, en el pasado este proceso seguía siendo manual, por lo que se buscó su automatización con tan sólo energizar el robot. Para ello se requiere un script automático *rc.local* [**42**], el cual ejecute todas las líneas de comando que se encuentren dentro de él en cuanto la Raspberry arranque el sistema operativo.

Para editarlo se usa el comando:
sudo nano /etc/rc.local

En este archivo se agrega la ejecución automática del script que contiene todos los preparativos para que ROS comience a funcionar, el cual se encuentra en */etc/rc.local* y puede consultarse en el Apéndice **J**.

Hay que recordar que es imperativo dejar la línea *exit 0* hasta abajo del documento para que no exista ningún problema. También, procurar evitar loops infinitos. Esto se logra agregando un ampersand (*&*) al final de las instrucciones.

7.8. Inhabilitación de permisos de administrador

Debido a que la ejecución del Shell Script de configuración automática elaborado en la sección anterior ayuda a darle permisos de escritura y lectura a los puertos de los módulos usados con ROS junto con la configuración del ROS MASTER para la visualización de Tópicos remota necesita ejecución de administrador con la instrucción **sudo**. Para ello se requirió editar el archivo *scratch* que se abre con la instrucción [59]:

sudo visudo

Este archivo se encuentra en el Apéndice **K**.

7.9. Pasos de ejecución para poner en marcha al Bípedo

Una vez que se tuvo todo listo, se preparó el robot para sus pruebas finales, las cuales resultaron exitosas. Los pasos que se deben seguir son los siguientes:

1. Energizar el robot mediante la fuente de 5[A]
2. Ingresar vía SSH con el equipo externo por medio de IP a la Raspberry con el comando: **ssh bipedoscout01@172.16.9.147**
3. Teclear **Ctrl + R** en la misma terminal y buscar el comando: **roslaunch Bipedo_Rasp Bipedo.launch**. Oprimir **ENTER**, o en su defecto, escribir la instrucción manualmente y oprimir **ENTER**

4. Abrir una terminal desde la computadora y corroborar que ya está conectado el robot con el comando: **rostopic list**. Se deben observar los Tópicos de publicación desde la Raspberry, entre ellos **/rosout** y **/rosout_agg**. Si no, consultar el Apéndice Ñ de posibles errores con solución

5. Ejecutar en el equipo externo el Nodo (como ejemplo) *control_postura_node.py* con el comando: **rosrun Bipedo_CIA control_postura_node**

6. Visualizar, controlar y desarrollar los algoritmos que se deseen apoyándose con las herramientas de la Sección **2.5** de conceptos básicos de ROS. Otra herramienta muy útil es **rqt_plot**, la cual ayuda a graficar los datos de los Tópicos publicados, pero se hablará de ella más adelante.

Capítulo 8

Análisis de resultados

Este capítulo describe analíticamente todos los resultados obtenidos a lo largo de las pruebas de control de postura, control de marcha y navegación autónoma, enfatizando en esta última la solución del dato de dirección/orientación necesario para realizar caminatas en curvaturas para comprobar experimentalmente las diferencias entre ambas arquitecturas: *BlackBoard* y *Peer-to-Peer*.

8.1. Evaluación cualitativa del Bípedo

En esta sección se comparan los resultados obtenidos por medio de la implementación del antiguo y del nuevo sistema. Dicha comparación del comportamiento del robot bípedo ocurre en dos circuntancias: con la corrección de ángulos en postura fija y otra en plena marcha. La evaluación está basada en la observación del Bípedo por un usuario externo y en la experiencia del mismo en la manipulación del robot.

8.1.1. Evaluación en postura fija

Realizando la comparación entre el sistema anterior y el nuevo sistema con ROS (Figura **8.1**) sobre la plataforma de pruebas del robot, ésta se modificó manualmente a **15 grados** de inclinación en aproximadamente **2 segundos**.

La respuesta en ambos sistemas fue prácticamente la misma ya que no se modificaron las leyes de control, sino la estructura de los procesos. Sin embargo, se realizó un experimento de inclinaciones libres y súbitas con el robot en el aire (Figura **8.2**) y, después de varias repeticiones en ambos, se detectó que con ROS existe un ligero incremento en el retraso de la respuesta, llegando

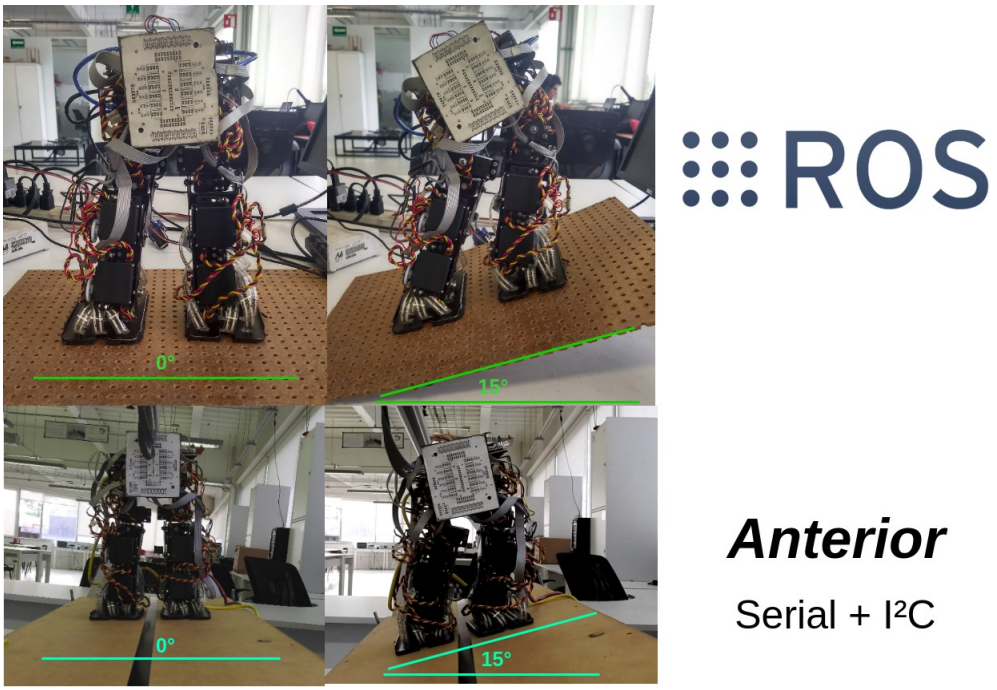

Figura 8.1: Experimento de inclinación a **15°** y corrección de postura con ROS en aproximadamente **2 segundos**

aproximadamente hasta **200 milisegundos**, a diferencia del anterior, que aparentaba ser de **100 milisegundos**.

Algunos posibles factores de tal aumento son:

- Apoyarse en el concepto de ROS MASTER y transmitirlo por medio de una IP hasta otro equipo causa que la transmisión de datos sea más costosa computacionalmente, ocasionando ligeros retrasos entre los paquetes de información. Además, a pesar de estar trabajando dentro de una red privada, varios usuarios tienen acceso a ella, lo que pudiera ocasionar lentitud en el tráfico de datos.

- ROS aprovecha el puerto Serial para cada dispositivo dependiendo de la capacidad del ancho de banda de la Raspberry. En este caso no hay muchos dispositivos, así que no existe demasiado retraso por dicho protocolo.

- Se presenta un retraso en la respuesta electromecánica de los servomotores. El desgaste electrónico de la tarjeta controladora (descalibración), junto con el juego mecánico de los engranes y el degrado del esmalte del embobinado del motor son causantes de pequeñas demoras que van amplificándose con el tiempo.

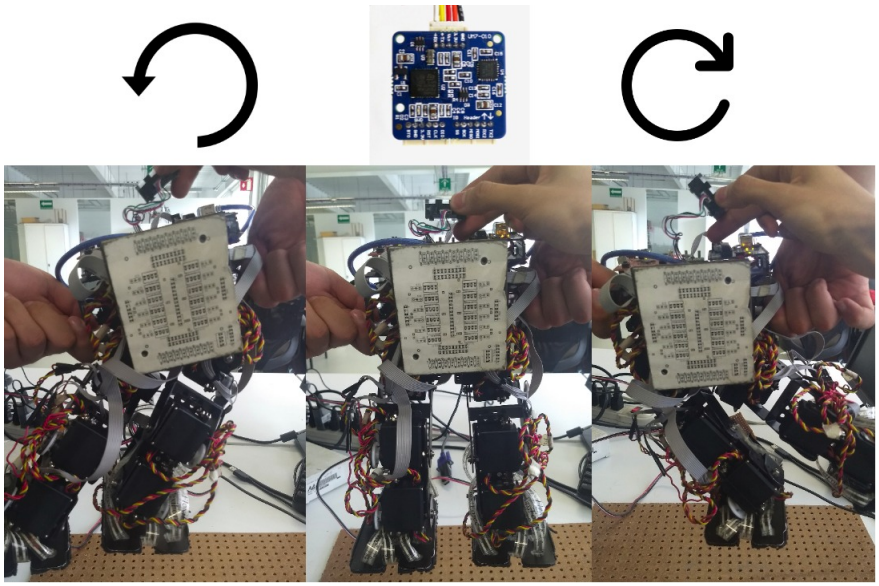

Figura 8.2: Experimento de inclinaciones libres y súbitas de ambos sistemas

8.1.2. Evaluación en plena marcha

Al igual que el experimento realizado en el desarrollo de controladores difusos [1], se ejecutó exitosamente la marcha bípeda del robot con ROS siguiendo una trayectoria recta establecida previamente en terreno plano totalmente horizontal y otro en inclinación (con límite de estabilidad de **8°**). Las Figuras **8.3** (superficie plana) y **8.4** (superficie inclinada) comparan ambas marchas bípedas.

Se puede apreciar un comportamiento muy similar en ambos. A pesar de tener un retraso ligeramente mayor en la respuesta del nuevo sistema, no repercute notablemente en la marcha bípeda. Sin embargo, fue conveniente saber con exactitud el retraso generado por el sistema con ROS, el cual se explica detalladamente en la siguiente sección.

8.2. Evaluación cuantitativa del Bípedo

La herramienta **rqt_plot** proporciona una Interfaz Gráfica de Usuario (GUI, por sus siglas en inglés) que permite visualizar valores numéricos en 2D acerca de los Tópicos publicados en ese momento como lo muestra el ejemplo de la ROS Wiki [58] de la Figura **8.5**.

Para aprovecharla (generalmente viene pre-instalada) es necesario hacer uso del siguiente comando una vez levantado el **roscore**:

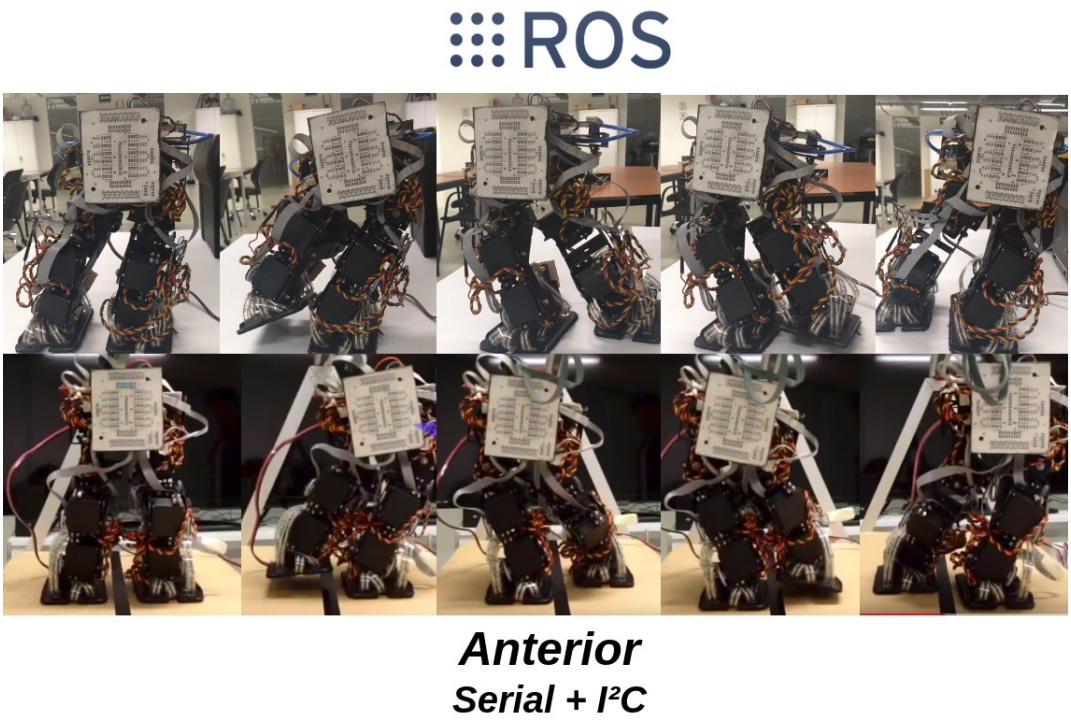

Figura 8.3: Comparación de respuesta en superficie plana de ambos sistemas [51]

rosrun rqt_plot rqt_plot

Gracias a la GUI se pueden analizar los datos de manera detallada y controlada, facilitando la visualización de estabilidad, niveles de discretización de curvas y pérdidas de datos.

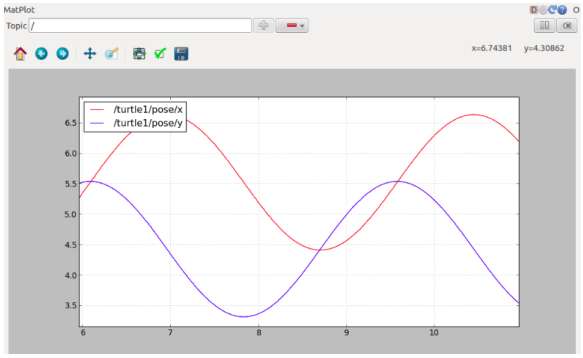

Figura 8.5: Uso de GUI **rqt_plot** para la visualización de Tópicos a través del tiempo [58]

8.2.1. Graficación del Control de Postura

Se planteó una estrategia de análisis para saber con exactitud qué es lo que estaba ocurriendo con los datos del robot a lo largo del tiempo en el control de postura, así que se comparó el dato de un único servomotor (con mayores variaciones en sus ángulos para una mejor observación)

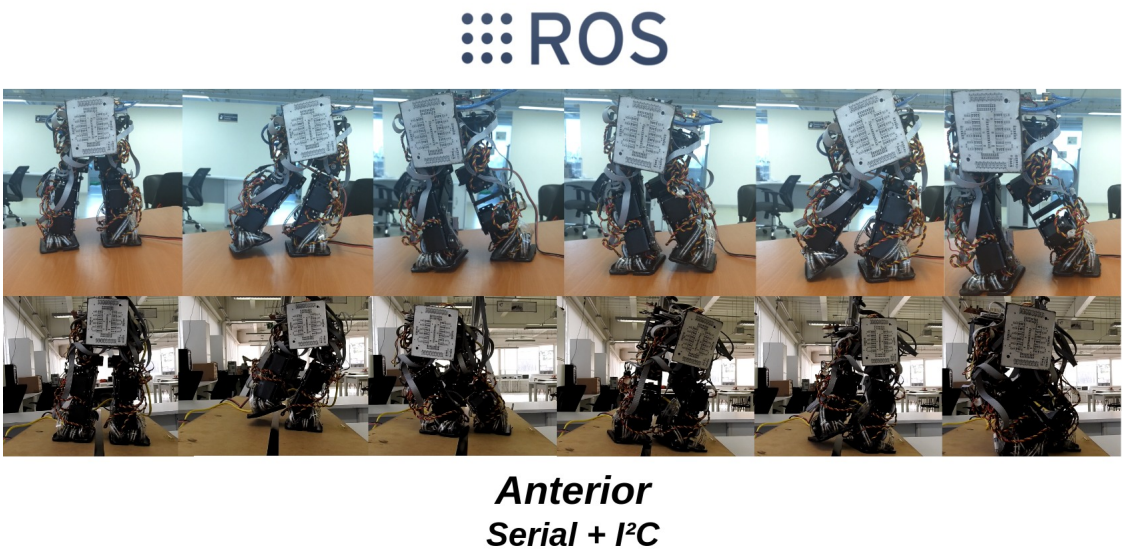

Figura 8.4: Comparación de respuesta en superficie inclinada a **8°** de ambos sistemas

desde el Tópico **servo_plot_compu/position/x** creado únicamente para fines gráficos, no de operación. Éste dato se transmite desde la computadora con el Nodo **control_postura_node.py** proveniente del Paquete **Bipedo_CIA**. Se realizaron pruebas de inclinación súbitas para tener el *peor escenario* en donde el robot tuviera que corregir su postura inmediatamente.

Dicha comparación fue hecha con el dato publicado del Tópico **servo_plot_tiva/position/x** creado también con propósitos gráficos. Proveniente desde el Nodo **tiva_servos_node.ino** de la Tiva, grafica **exactamente** el mismo dato calculado del servomotor usándolo como comprobación de las instrucciones a escribir en el actuador.

Se compararon las gráficas de lo que el robot tenía que hacer, contra lo que realmente informaba que hacía.

Como se puede observar en la Figura **8.6** existe un retraso notable entre ambas curvas. A simple vista, se afirma que el dato de la Tiva se obtuvo con un mayor tiempo de muestreo, ocasionando pérdidas en la información, a pesar de que sean los mismos valores en ambos Tópicos.

Realizando un acercamiento a este comportamiento en la Figura **8.7**, se observa que existe una sección con retraso de **269 milisegundos** entre los datos calculados de la computadora y los datos escritos en la Tiva para el servomotor. En dicho intervalo, hubo **4 cambios** de valores y **1 cambio** en la escritura, teniendo una pérdida total de **3 valores** con una diferencia de **5°**.

Para saber cual era el rango máximo de variación de los servomotores, se graficaron todos ellos con la herramienta **rqt_plot**. Se halló que con las leyes de control realizadas en los trabajos

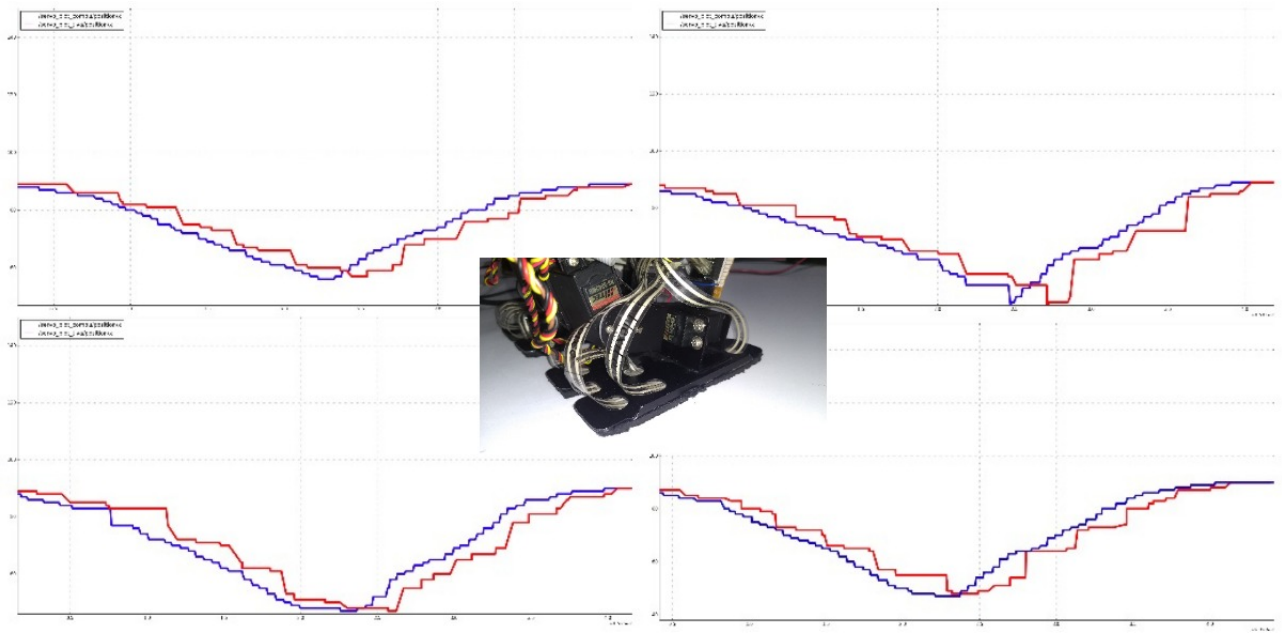

Figura 8.6: Movimiento de servomotor **11** (pie izquierdo) en control de postura rápida con inclinación derecha en **1 segundo**. Laptop (**azul**), Tiva-C (**rojo**)

anteriores se alcanza un máximo de **74°** de variación en los ángulos (de 180° disponibles) como se muestra en la Figura **8.16**.

Mediante un experimento en donde se modificó temporalmente el código de control en el sistema anterior, se obtuvo experimentalmente que el robot comienza a tener dificultades para ejecutar correctamente la marcha bípeda a partir de **330 milisegundos** de retraso y **11°** de error absoluto (continuo en casi todos los datos) en la corrección de postura, equivalente al **14.86 %** de error relativo mediante la ecuación **8.1** [**52**], donde x es el valor experimental y x0 es el valor esperado (74°). El resultado se le resta a 100 ya que x0 no es exactamente un valor deseado, es una referencia.

Con base a la prueba anterior, se determinó que el *peor escenario* de inclinaciones súbitas mediante el Control de Postura con ROS tuvo un error relativo del **6.76 %** dentro del segmento de **269 milisegundos**.

$$\epsilon = 100 - |(x - x_0)|/x_0 * 100[\,\%] \tag{8.1}$$

Ambas gráficas de la Figura **8.7** alcanzan los máximos y mínimos por igual debido a que son exactamente el mismo dato. Por esta misma razón, no existe sobrepaso y la forma general de la curva original se mantiene.

El caso de la inclinación izquierda no fue diferente, la Figura **8.8** muestra dos secciones de interés. La primera presentó un desfase de **305 milisegundos** y **3** pérdidas de datos con error

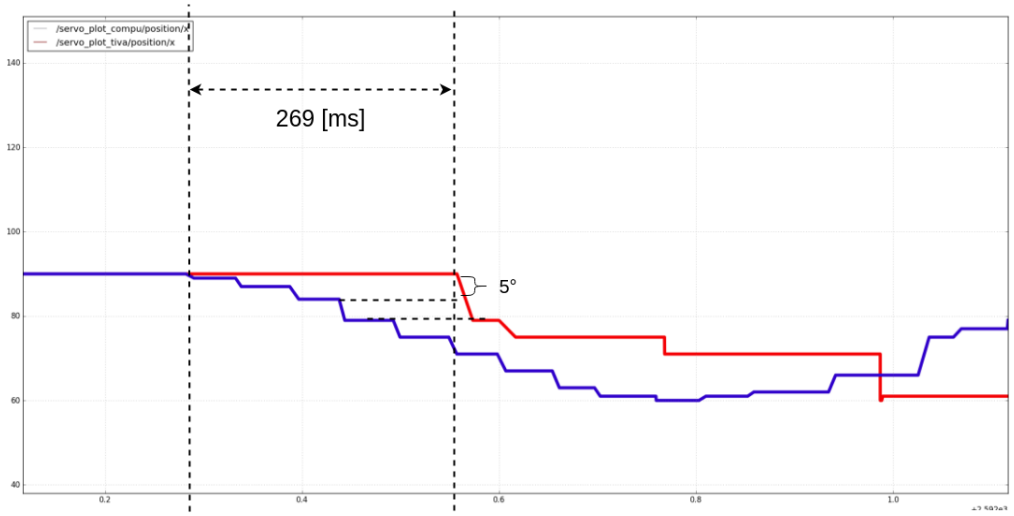

Figura 8.7: Acercamiento de movimiento del servomotor **11** (pie izquierdo) en control de postura rápida con inclinación derecha en **1 segundo**. Laptop (**azul**), Tiva-C (**rojo**)

absoluto de **12°**, lo que equivale a un error relativo del **16.21 %** calculándolo con la ecuación **8.1**. La segunda tuvo un desfase de **129 milisegundos** y **ninguna** pérdida, lo cual significa **0.00 %** de error relativo.

El principal objetivo de la variación súbita de ángulos durante el Control de Postura era determinar a mayor detalle el desfase entre las gráficas de lo que **se le manda** al robot y **lo que el robot informa que ejecuta**, tomando con poca relevancia a los errores relativos para la aplicación del Control de Marcha (ya que son hipotéticos). No se realizó esta acción inicialmente en la marcha bípeda ya que, al tener muy pocas variaciones en los datos, se dificultó mucho la determinación numérica del retraso entre las curvas, por lo que se prefirió realizar el experimento primero con el Control de Postura.

Para entender mejor el comportamiento del sistema, se registraron aleatoriamene 22 secciones de varias gráficas similares (incluyendo las de las Figuras **8.7** y **8.8**) en la Tabla **8.1**. Se observó de inmediato que había una relación directa entre el tiempo de retraso (desfase) y el número de datos perdidos. Sin embargo, no existía dicha proporción entre el error relativo de ángulos y el desfase entre curvas, ya que la cantidad de error dependía del tipo de movimiento del robot, no del tiempo de retraso entre las curvas.

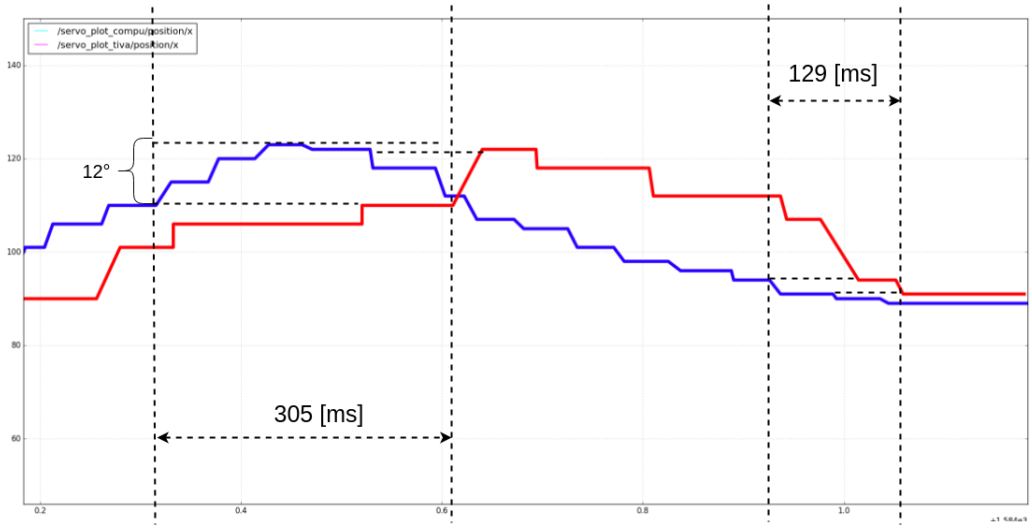

Figura 8.8: Acercamiento de movimiento del servomotor **11** (pie izquierdo) en control de postura rápida con inclinación izquierda en **1 segundo**. Laptop (**azul**), Tiva-C (**rojo**)

Tabla 8.1: Datos experimentales de desfase de curvas y datos perdidos

	Desfase [ms] (x)	**Datos perdidos [n] (y)**	**Error absoluto [°]**	**Error relativo [%]**
1	84	0	0	0.00
2	88	0	0	0.00
3	103	0	0	0.00
4	129	0	0	0.00
5	137	0	0	0.00
6	142	0	0	0.00
7	180	1	1	1.35
8	183	1	2	2.70
9	221	2	2	2.70
10	235	2	2	2.70
11	243	2	6	8.10
12	246	2	3	4.05
13	251	2	4	5.40
14	259	2	2	2.70
15	269	3	5	6.76
16	275	3	5	6.76
17	289	3	4	5.40
18	292	3	4	5.40
19	305	3	12	16.22
20	315	4	15	20.27
21	320	4	4	5.41
22	338	5	14	18.92

Para obtener el valor aproximado en el cual el sistema comienza a tener pérdidas de datos se realizó una regresión lineal mediante el método de mínimos cuadrados [53]. Sólo se tomaron en

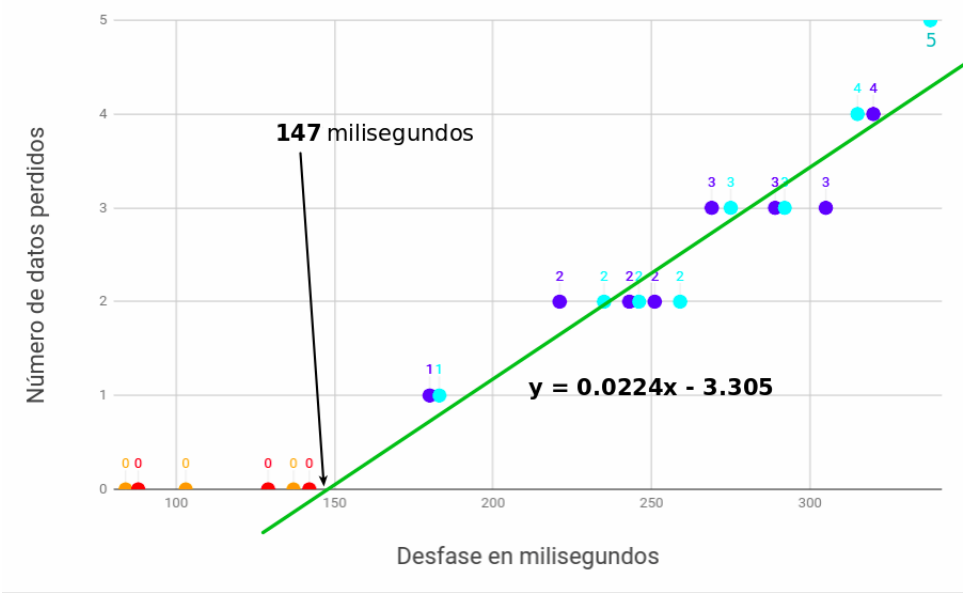

Figura 8.9: Regresión lineal de gráfica de desfase entre curvas [ms] contra error relativo de sincronización de datos [n].

cuenta los datos con pérdidas de valores diferentes de 0 en la expresión **8.2**, dando como resultado la ecuación de la recta **8.3**. Al realizar su intersección con el eje x (y = 0), se encuentra que el valor aproximado en el cual el sistema puede comenzar a presentar pérdidas de datos es **147 milisegundos**.

$$y = \frac{n\sum(xy) - \sum x \sum y}{n\sum x^2 - |\sum x|^2}x + \frac{\sum y \sum x^{-} \sum x \sum(xy)}{n\sum x^2 - |\sum x|^2} \tag{8.2}$$

$$y = \frac{10(8999) - 30(2731)}{10(110) - (30)^2}x + \frac{2731(110) - 30(8999)}{10(110) - (30)^2}$$

$$\mathbf{y = 0.0224x\text{-}3.305} \tag{8.3}$$

La primer columna de la tabla **8.1** fue utilizada para calcular una curva de Distribución Normal o bien conocida como Campana de Gauss, la cual es una representación gráfica del Teorema de Distribución Normal. Ben Collins, analista de datos [54] enuncia:

"*Los datos convergen alrededor de la media (promedio) sin sesgo hacia la izquierda o hacia la derecha. Eso significa que se puede saber la probabilidad de cuántos valores ocurrieron cerca de la media.*"

Figura 8.10: Gráfica de Distribución Normal o Campana de Gauss del tiempo de retraso

$$\mu = \frac{\sum x}{n} = \frac{4221}{22} = 223 \tag{8.4}$$

$$\sigma = \sqrt{\frac{\sum (x-\mu)^2}{n-1}} = \sqrt{\frac{133104}{21}} = 79{,}61 \tag{8.5}$$

Mediante el cálculo de la media o promedio (Ecuación 8.4) y el cálculo de la desviación estándar (Ecuación 8.5), donde x es cada dato de desfase entre las curvas y n es el número de muestras, se pudo determinar que la probabilidad de manifestación del rango [143,305] milisegundos es del 68.3 %. En otras palabras, sumando y restando una desviación estándar a la media, aproximadamente el **70 %** de las veces pueden llegar a presentarse valores de retraso entre **143** y **305 milisegundos** con pérdidas de **0** hasta **3 datos**.

¿De qué manera afecta dicho rango? Previamente se había encontrado que arriba del **14.86 %** de error relativo (continuo) entre ángulos comenzaba a afectar de manera importante al robot. Es cierto que en la Tabla 8.1 se registró un dato de **20.27 %**, pero dicho dato no entra en el **70 %** de las ocasiones, catalogándolo como un dato *atípico*. Además, dentro del rango de retraso únicamente se presentó una sola vez un **16.22 %** de error, mientras que las demás muestras siempre se mantuvieron debajo del **8.11 %**. Con esto se comprobó que el nuevo sistema no repercute de manera importante si se presentan errores cercanos al **14.86 %**, siempre y cuando sean datos esporádicos.

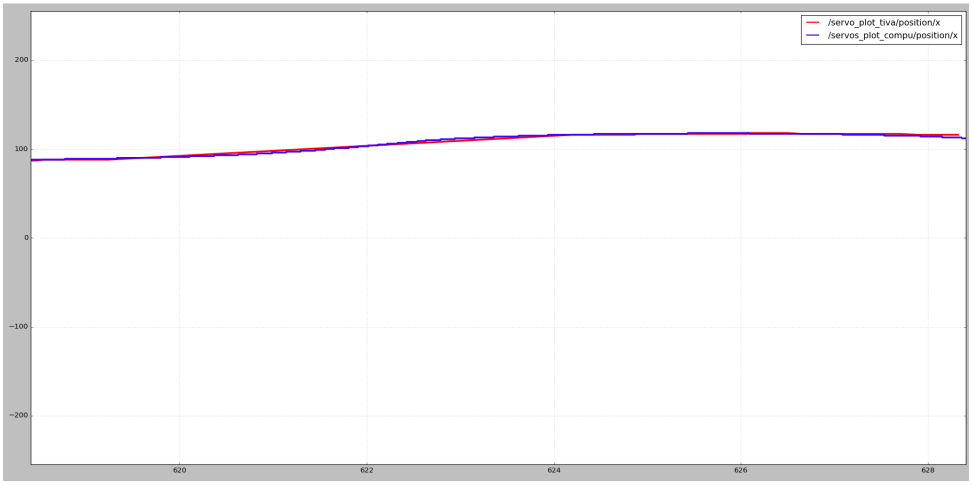

Figura 8.11: Acercamiento de movimiento del servomotor **11** (pie izquierdo) en control de marcha en **10 segundos**. Laptop (**azul**), Tiva-C (**rojo**)

8.2.2. Graficación del Control de Marcha

La Figura **8.11** muestra el comportamiento del mismo servomotor usado para fines gráficos, pero ahora ejecutando el Control de Marcha en **10 segundos**.

Realizando un acercamiento de la imagen, se puede observar en la Figura **8.12** que, a pesar de tener el mismo tiempo de muestreo que en la prueba de inclinación súbita en el Control de Postura (**1 segundo**), el desfase entre ellas es mínimo, presentándose en esta ocasión como **130 milisegundos**. Esto puede deberse a que la cantidad de información mandada en el Control de Marcha es más *digerible* para los Tópicos, permitiendo sincronizar ambas curvas en un menor periodo de tiempo.

Con referencia a la Tabla **8.1** y la regresión lineal de la Figura **8.9**, se puede afirmar que los valores de desfase inferiores a **147 milisegundos** no presentarán ninguna pérdida de datos, que es el caso de la Figura **8.12**. Tampoco se registraron pérdidas de información en todos los 10 segundos de marcha bípeda.

Tomando en cuenta que no hubo dichas pérdidas y que es de interés conocer la diferencia de ángulos máxima, el robot presentó en la sección de los **130 milisegundos** un error absoluto de **2°**, teniendo éste un error relativo del **2.70 %** que no se acerca ni un poco al límite del **14.86 %**. El dato más grande que se alcanzó a registrar a lo largo de toda la curva fue de **5°** con **6.76 %** de error relativo, pero sigue estando dentro de la tolerancia.

Como nota final de esta sección, se especuló que al tener un suscriptor y un publicador en el mismo microcontrolador (Tiva-C o Arduino UNO) se genera una desincronización a lo largo del

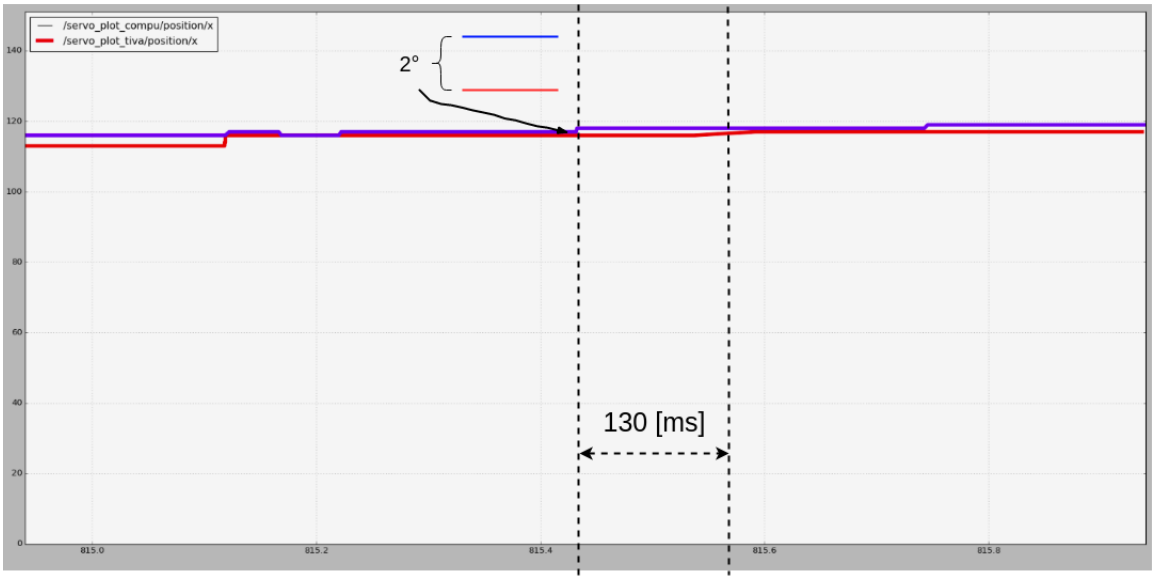

Figura 8.12: Acercamiento de movimiento del servomotor **11** (pie izquierdo) en control de marcha en **1 segundo**. Laptop (**azul**), Tiva-C (**rojo**)

tiempo y este mismo efecto es el causante del retraso en la gráfica de escritura de los servomotores, por lo que se decidió quitar el Tópico publicador (que sirve para fines de graficación) después de las pruebas experimentales. Posterior a ello, el robot comenzó a responder a mayor velocidad y con mayor estabilidad. **Posiblemente, el retraso generado sea reacción del uso simultáneo de un suscriptor y publicador en un microcontrolador**, causando pérdidas de datos y retraso en las curvas; éste no es el único proyecto en donde se ha observado lo anterior.

8.2.3. Procesamiento

Scout Realtime (ejemplo en la Figura **8.13**) es una herramienta de monitoreo de servidor web simple y de uso amigable, que monitorea las métricas de Linux en tiempo real. Análogo a la herramienta **top**, Scout Realtime permite visualizar de manera gráfica el consumo de procesamiento del robot, siendo definido por su página oficial como: *El **top** del desarrollador moderno* [55].

Las métricas de interés en **scout_realtime** básicamente son dos:

- **Procesamiento de CPU (Unidad Central de Procesamiento, por sus siglas en inglés)**: Es la encargada de interpretar instrucciones de un programa a través de las operaciones aritméticas, lógicas, así como de entrada y salida del sistema, comenzando a alentarse aproximadamente a partir del 70 %.

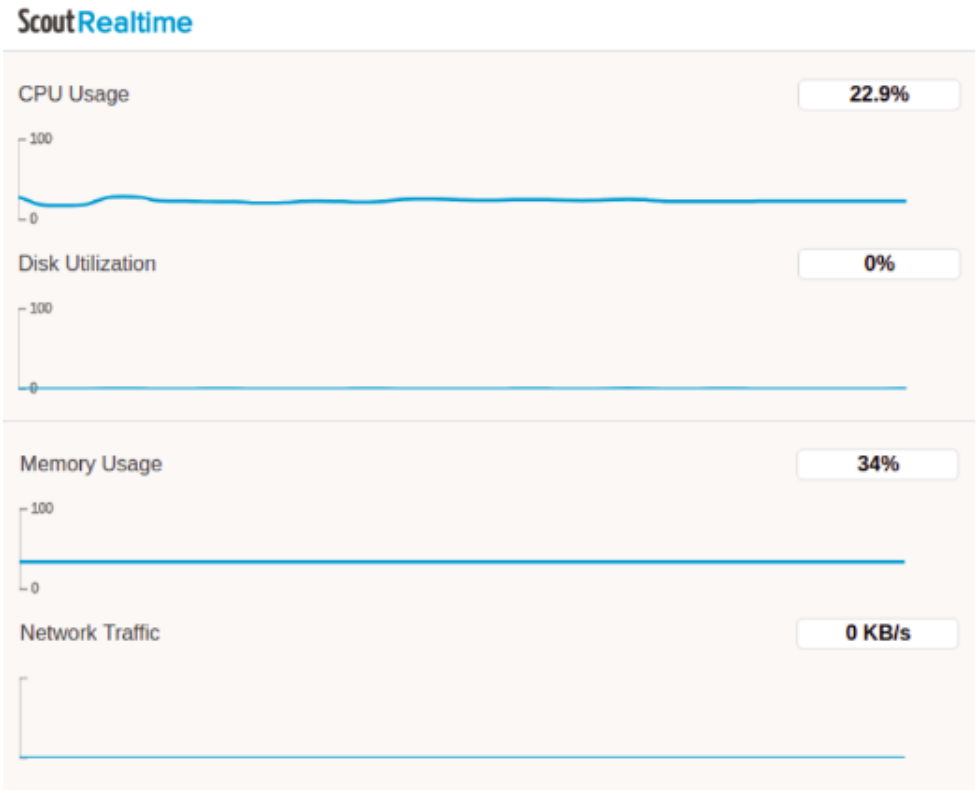

Figura 8.13: Monitoreo de métricas de procesamiento de la Raspberry por medio de Scout Realtime

- **Consumo de MEM (Memoria)**: Es utilizada como memoria de trabajo. En ella se cargan todas las instrucciones que ejecuta la CPU, así como otras unidades. De igual manera, hay que prestar atención una vez alcanzado el 70 % de su consumo.

Nuevo sistema: con ROS

El uso de CPU y Memoria de la Raspberry sin ningún proceso del robot fue de **3 %** y **28 %** respectivamente.

Después de ejecutar el **roslaunch**, que levanta tanto el ROS MASTER como la comunicación serial con el Arduino, la Tiva y la UM7, el consumo de recursos subió a **22.9 %** en CPU y a **34 %** en Memoria. Sin embargo, al ejecutar el control de postura y el control de marcha, no hubo ninguna variación de estos porcentajes en la Raspberry. Esto se debe a que los Nodos de control se encontraban en un equipo externo, permitiendo al *cerebro* principal del robot tener menos carga de trabajo.

Utilizando el nuevo sistema con ROS, el uso de CPU y Memoria del equipo externo sin ningún proceso del robot fue de **7.19 %** y **17 %** respectivamente, sin tomar en cuenta lo que **chrome** consume (necesario para abrir Scout Realtime).

Tabla 8.2: Consumo de procesamiento y memoria **con ROS**

Proceso ejecutado - IMG 16GB	**CPU Raspberry Pi3**	**MEM Raspberry Pi3**	**CPU HP Gaming Pavilion 15**	**MEM HP Gaming Pavilion 15**
Ninguno	**3 %**	**28 %**	7.19 %	17 %
roslaunch: ROS MASTER, conexión Tiva-C, Arduino y UM7	**22.9 %**	**34 %**	7.19 %	17 %
Control de Postura	**22.9 %**	**34 %**	22.92 %	18 %
Control de Marcha	**22.9 %**	**34 %**	15.51 %	19 %

Tabla 8.3: 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 %**

Después de ejecutar el Nodo de Control de Postura, aumentó el uso de CPU a **22.92 %** y el uso de Memoria a **18 %**.

Y, finalmente, al ejecutar el Nodo de Control de Marcha, el uso de CPU llegó a **15.51 %** y el uso de Memoria a **19 %**. Los consumos de procesamiento y memoria del nuevo sistema se muestran en la Tabla **8.2**.

Antiguo sistema: sin ROS

Se realizó el mismo método de medición de parámetros con Scout Realtime. El uso del CPU y Memoria de la Raspberry sin ningún proceso del robot fue de **4 %** y **13 %** respectivamente.

Ejecutando directamente con el comando **python** el Nodo Control de Postura, el uso del CPU y Memoria de la Raspberry fue de **37.7 %** y **17 %**, y con el Control de Marcha, de **44.7 %** y **18 %** respectivamente. Los consumos de procesamiento y memoria del antiguo sistema se muestran en la Tabla **8.3**.

La Figura **8.14** muestra la comparación entre ambos sistemas considerando el procesamiento de CPU y Memoria del robot. Nótese que **roslaunch** no aplica para el sistema anterior, por lo que se mantiene en el mismo procentaje.

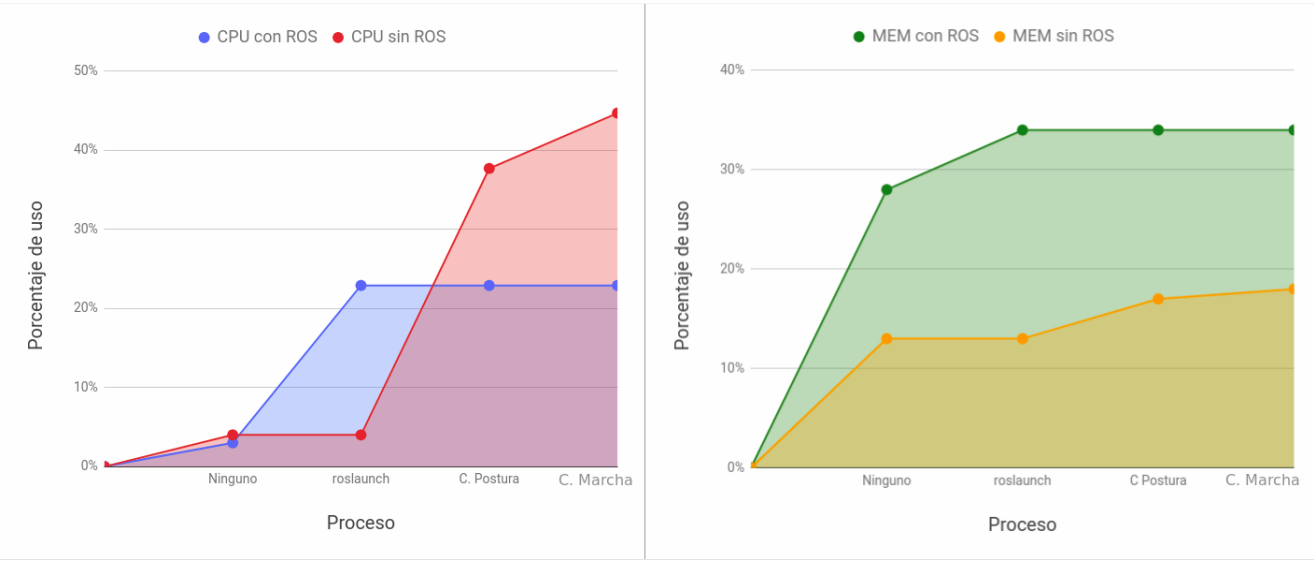

Figura 8.14: Comparación entre ambos sistemas en procesamiento de CPU y MEM

Claramente se puede ver que el sistema anterior sin ROS consume más CPU, pero menos Memoria, esto debido principalmente a que se destinó la ejecución de los algoritmos de control a un equipo externo en vez de saturar a la Raspberry.

A pesar de que ROS consume más recursos de Memoria, permite el fácil intercambio de datos con una computadora externa, ayudando a disminuir dramáticamente las cargas de trabajo en los núcleos principales.

¿Qué es más importante, la CPU o la Memoria?

Ambos lo son, pero la **CPU tiene mayor importancia** ya que representa la potencia del procesamiento. Además, es más sencillo aumentar la capacidad de memoria RAM que sustituir el procesador por otro con mejores características.

Se puede realizar la siguiente analogía entre la comparación de dos máquinas:

Máquina A: 1.3 GHz Pentium 4 con 4 GB de RAM
Máquina B: 2.2 GHz Core 2 Duo con 2 GB de RAM

La Máquina A podrá tener el doble de memoria que la B, pero será mucho más lenta ya que tiene 1 único núcleo a menor frecuencia, sin mencionar que será incapaz de utilizar los 4 GB de RAM por la misma razón. En cambio, la Máquina B será notablemente más rápida porque incluye 2 núcleos más veloces y gestionará de mejor manera la RAM, a pesar de ser de menor capacidad que la de la Máquina A. **Incluso aunque ambas tuvieran la misma memoria, la Máquina**

B con dos núcleos seguiría trabajando mejor [56].

Teniendo en cuenta lo anterior y que se trabajó con la Raspberry Pi3 Modelo B con 4 núcleos a 1200MHz y 1 GB de RAM, se puede asegurar que es más favorable ahorrar procesamiento en CPU (como lo que se realizó con ROS) a pesar de tener un mayor consumo de memoria RAM. En ambos sistemas ninguno llega a niveles críticos (arriba del **70 %** u **80 %**, ni en CPU, ni en MEM), pero para propósitos futuros está numéricamente demostrado que ROS aportará herramientas con mejor oportunidad de desarrollo robótico.

8.2.4. Estabilidad computacional

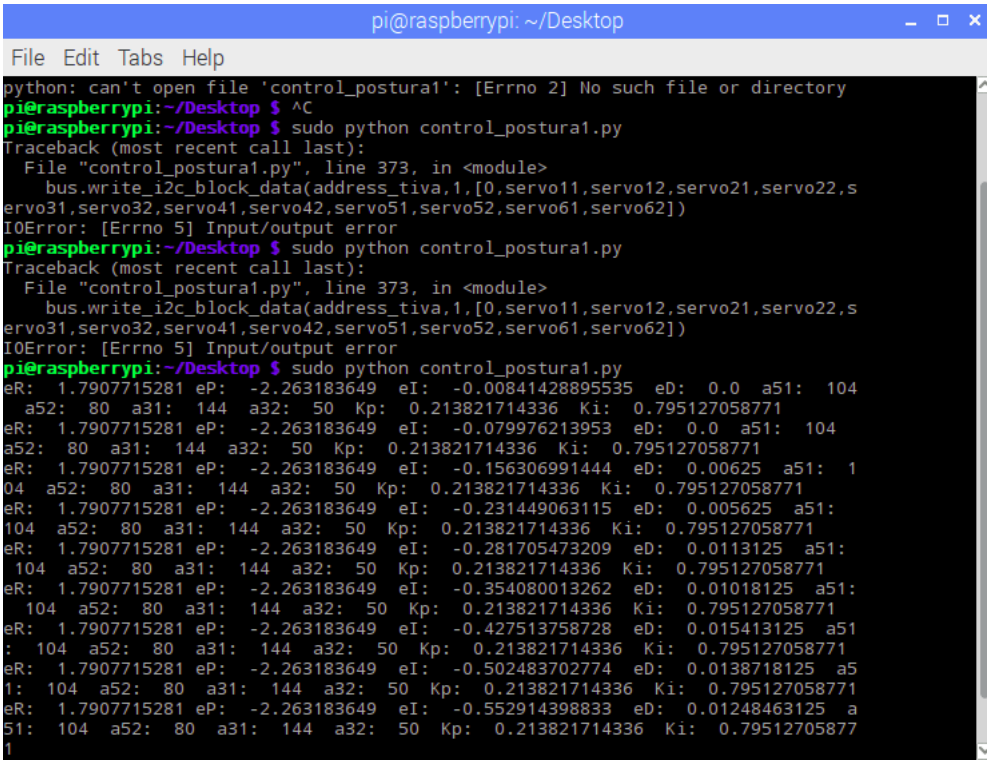

Figura 8.15: Error de entrada/salida referente al puerto I²C utilizado en la comunicación de la Tiva-C. Vinculación exitosa hasta el tercer intento

La Figura **8.15** muestra la terminal única de visualización del proceso del control de postura del sistema anterior, en donde después de realizar de **15 a 20 intentos**, surgieron los distintos tipos de error:

1. **Checksum error**, deteniendo completamente el robot

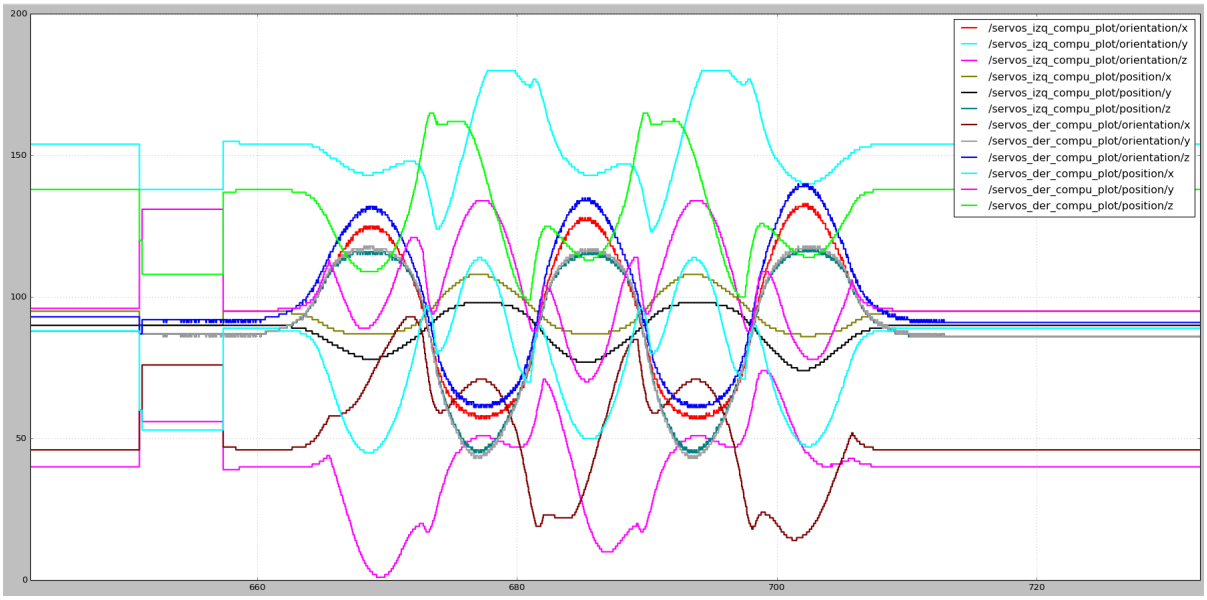

Figura 8.16: Graficación de los **12** servomotores en conjunto de toda la marcha bípeda en **63 segundos** con una variación total de **74°**

2. **Pérdida de comunicación serial con la UM7** provocada por falsos contactos, desincronización, o mala inicialización del dispositivo

3. **Pérdida de datos de la UM7**, sin romper totalmente la comunicación serial

4. **Error de entrada/salida en el puerto I²C con la Tiva-C**, vinculándose hasta el tercer intento

El Nodo de Control de Postura es un bucle infinito y el Control de Marcha dura **62 segundos**, por lo que se puede afirmar que el umbral requerido exije **0 fallos**, por lo menos, en los primeros **63 segundos**.

La Figura **8.16** muestra el comportamiento de los **12** servomotores a lo largo de la marcha bípeda en **63 segundos**. Se puede observar claramente que con ROS ya no existe ninguna pérdida de datos dentro de este rango de tiempo debido a la continuidad de sus curvas, permitiendo cálculo y escritura de los valores de los servomotores por medio de PWM.

La misma continuidad se presentó con los valores de Roll, Pitch, Yaw, los potenciómetros y los sensores Flex, pero debido a su desgaste en trabajos previos, no se pudieron obtener al **100 %** los **33** valores proporcionados de los sensores vía serial por el Arduino.

8.3. Implementación de solución para el ángulo de dirección/orientación

La IMU UM7, como ya se mencionó en la Sección **4.2**, fue una excelente elección para las mediciones de ángulos de inclinación. Sin embargo, no fue el mejor módulo para la búsqueda del ángulo de orientación al presentarse perturbaciones (señales indeseadas) específicamente en este eje. Para poderlo obtener se requirió hacer uso del magnetómetro del dispositivo (Tópico **imu/mag**), el cual funciona tomando en cuenta los campos magnéticos de la Tierra y, así, obteniendo la orientación hacia el norte (Azimuth), representando la orientación absoluta. La dirección relativa (Yaw) también se obtuvo, pero con el Tópico **imu/rpy**. Sus diferencias pueden consultarse más a detalle en la Sección **2.7**.

Además de comportarse de manera distinta, el eje Azimuth y el eje Yaw de la UM7 nunca proporcionaron los resultados requeridos para la aplicación del robot bípedo desde las tesis anteriores.

Gracias a la experiencia obtenida a lo largo de la carrera de mecatrónica, se optó por usar la IMU MPU-6050 (explicada en la Sección **4.5**) como módulo de obtención del ángulo de dirección en lugar de invertir tiempo y esfuerzo en el filtrado de la señal de la UM7.

8.3.1. Prueba estática comparativa con magnetómetro (Azimuth)

Se realizaron varias pruebas muestreadas a **1 segundo**, donde cada IMU tenía un origen diferente, una por su propia calibración relativa (MPU-6050) y otra por la supuesta indicación al norte (UM7).

La primera prueba tuvo como características un origen relativo en **6.8°** para la MPU-6050 y un origen absoluto en **12.5°** con respecto al Norte para la UM7 (que sí es un magnetómetro). El tenerlas desfasadas ayudó a observar las curvas de manera más detallada.

Usando la herramienta **rqt_plot** (ya explicada en la Sección **8.2**) se puede observar claramente que en la Figura **8.17** la curva de dirección de la MPU-6050 (**verde**) es mucho más estable y lineal que la de orientación de la UM7 (**naranja**).

El magnetómetro tuvo un valor máximo de **13.2°** y un mínimo de **9.9°**, presentando un error relativo del **12.5 %**. En cambio, el filtro complementario de la MPU-6050 se mantuvo exactamente en la misma posición en toda su curva por el trabajo colaborativo del acelerómetro y giroscopio. Se observaron algunas variaciones en los datos de la MPU-6050 con la herramienta **rostopic echo** en el orden de centésimas, estimándose un error relativo menor a **0.1 %**, representando éste una magnitud de falla despreciable para el robot.

La UM7 presentó en total **16 cambios** de ángulos en el tiempo transcurrido de la prueba. La

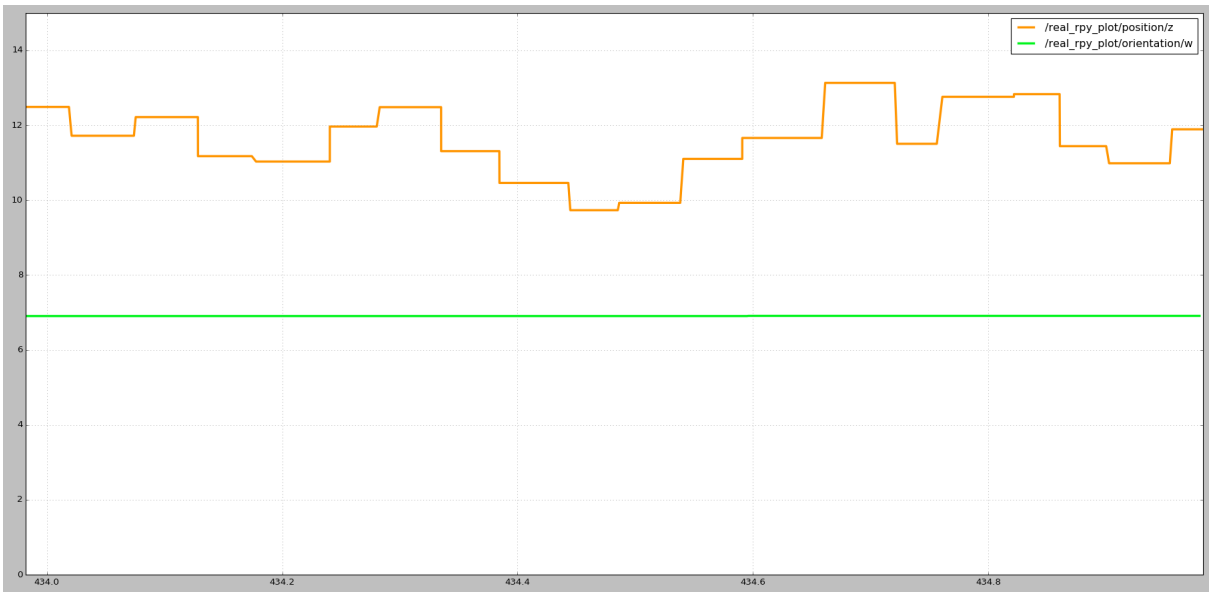

Figura 8.17: Primera comparación de comportamiento estático del ángulo de orientación en 1 segundo de la MPU-6050 (**verde**) contra UM7 (**naranja**)

MPU-6050, **ninguno**.

Se observó que el magnetómetro en cada prueba tenía diferente origen al mover el robot y regresarlo a su posición original (lo cual no debía de suceder), permitiendo con eso saber que **el objetivo principal del magnetómetro no se lograba nunca**. Teniendo en cuenta lo anterior, no tuvo caso seguir estudiando el comportamiento del magnetómetro, así que se concluyó el experimento y se procedió a la comparación del eje Yaw del MPU-6050 con el de la UM7.

8.3.2. Prueba estática comparativa con Yaw (RPY)

La Figura **8.18** muestra el comportamiento de los datos de Yaw de ambas IMU, siendo la de la MPU-6050 la **verde** y la de la UM7 la **morada**.

Se detectó otro tipo de problema en el eje Yaw de la UM7 a diferencia de su magnetómetro. Esta vez no generó demasiado ruido, pero presentó un aumento de su valor en un corto periodo de tiempo.

Siendo más precisos, esta IMU presenta un incremento de **27°** a **39°** en **1 segundo**, es decir, una variación total de **12°** en **1 segundo**.

Al desplazar un poco la UM7, el fenómeno anterior no permitió saber si realmente se estaba siguiendo o no el giro total del robot, por lo que se concluyó que tampoco tenía caso continuar estudiando su comportamiento y se procedió a realizar la prueba colaborativa de ambas IMU

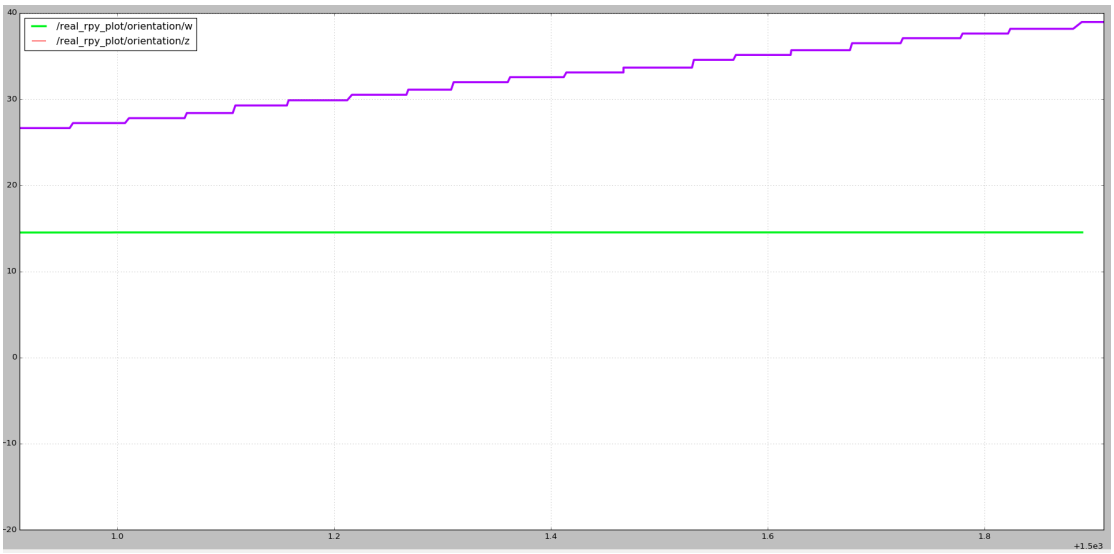

Figura 8.18: Comparativa de estabilidad estática de datos en **1 segundo** del eje Yaw del MPU-6050 (**verde**) y UM7 (**morada**)

combinando los ejes funcionales Roll y Pitch de la UM7 junto con el eje Yaw de la MPU-6050, **obteniendo finalmente el RPY real del robot**.

8.3.3. Prueba estática MPU-6050 + UM7: RPY real

Ahora se sabe que los dos primeros ejes de la UM7 (Roll y Pitch) funcionan correctamente para las leyes de control de postura y marcha, y que el eje de la MPU-6050 (Yaw) sirve de manera excelente para alguna futura ley de control para caminatas en curvaturas.

En conjunto, los tres ejes funcionales de ambos dispositivos se muestran en la Figura **8.19**, la cual muestra una variación de datos **nula** a lo largo de **1 segundo** de prueba.

8.3.4. Prueba dinámica del conjunto RPY

La Figura **8.20** muestra la prueba de variación de datos positivos y negativos en **1 segundo** de muestreo en Roll, donde se puede observar que, al ser una acción manual, varían los tres ejes, pero en especial los ángulos en Roll (**rojo**).

Se realizó lo mismo con Pitch (**verde**), que se muestra en la Figura **8.21**. Se puede observar claramente cómo la curva de interés sobresalta en variación con respecto a las otras dos.

Por último, se aplicó la misma metodología para el eje Yaw (**azul**), donde se muestran sus respectivos movimientos en las gráficas de la Figura **8.22**. Además, se observó que a diferencia del

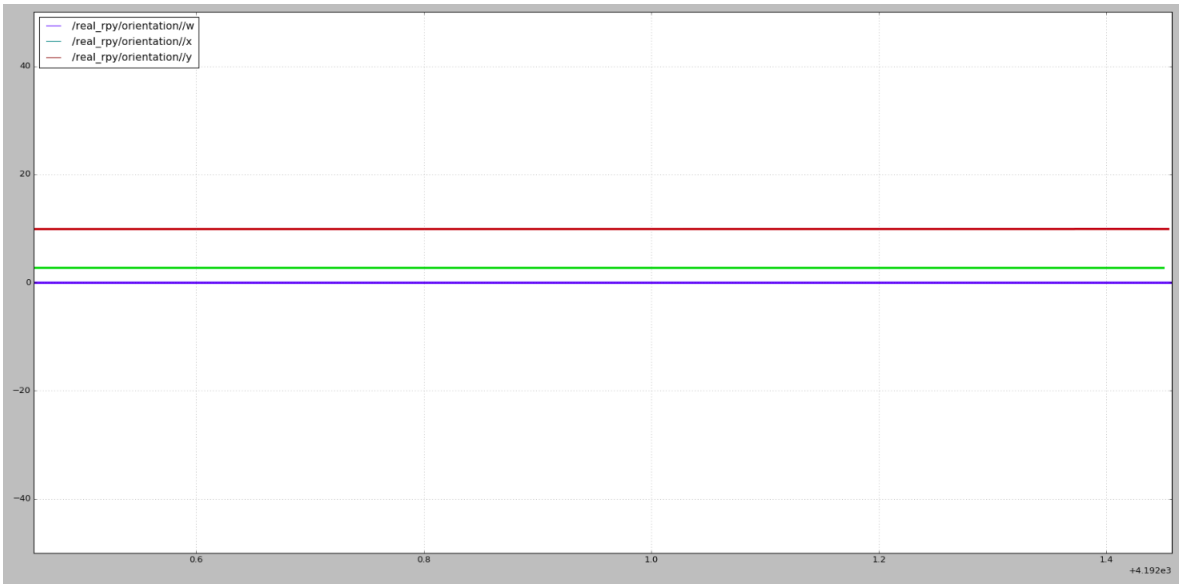

Figura 8.19: Colaboración de dispositivos con los ejes Roll y Pitch de la UM junto con el eje Yaw de la MPU-6050

magnetómetro y del eje Yaw de la UM7 (que jamás funcionaron correctamente), el eje Yaw de la MPU-6050 sí regresaba a su origen al posicionarlo en su postura de inicio.

8.4. Comparativa de soporte y desarrollo

¿Cómo saber qué tan útil es el soporte en el desarrollo de futuros proyectos para este robot?

Para saber la respuesta se propuso aprovechar el Tópico **/yaw** publicado desde el Nodo **sensores_node.py** proveniente del Paquete **Bipedo** que se encuentra dentro de la Raspberry para iniciar la introducción a la Navegación Autónoma como posible trabajo a futuro.

8.4.1. Nodo de Navegación Autónoma

Este Nodo tiene como objetivo principal demostrar que ROS es una excelente herramienta de desarrollo de proyectos en conjunto, combinando los tres conceptos:

1. Seguimiento de trayectorias de marcha bípeda y control PID autosintonizable [1]

2. Corrección de postura a lo largo de la marcha [1]

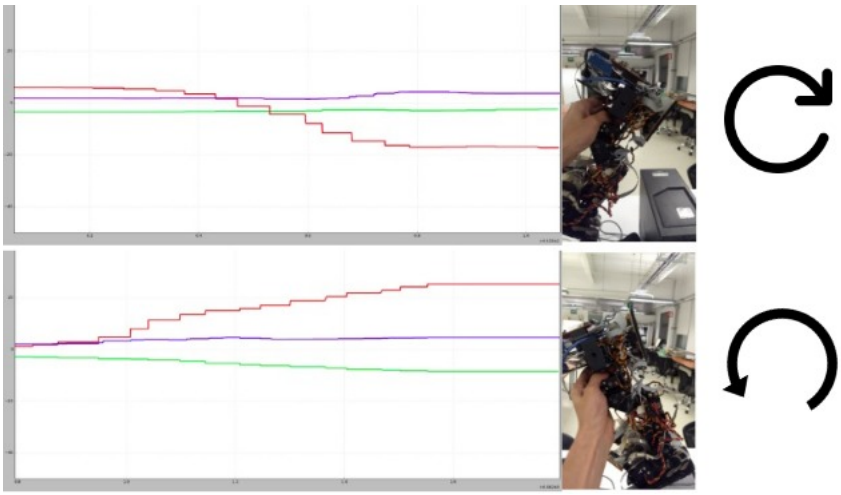

Figura 8.20: Prueba gráfica de variación de datos en Roll proveniente de la UM7 (**rojo**)

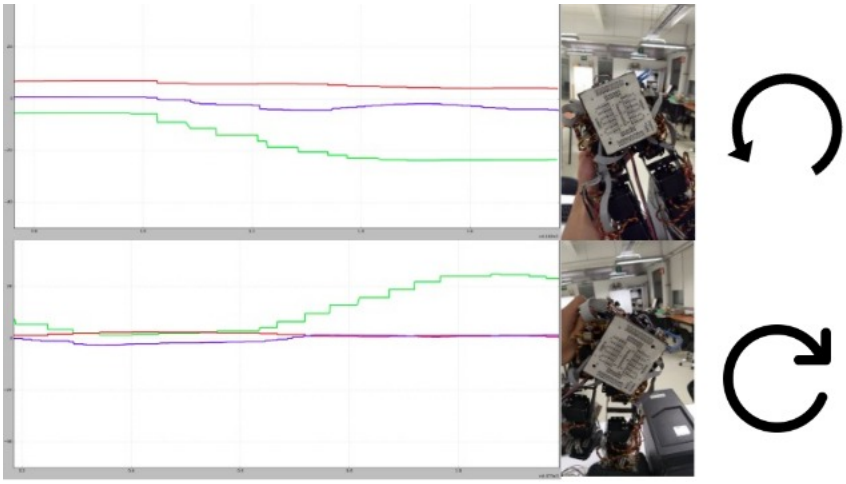

Figura 8.21: Prueba gráfica de variación de datos en Pitch proveniente de la UM7 (**verde**)

3. Control proporcional para seguimiento de un ángulo objetivo de dirección por medio de la marcha bípeda en curvaturas

Se entiende por navegación autónoma en robótica a *la capacidad de un robot de ir de un punto del espacio a otro, interactuando con su entorno* [57]. En el caso específico de este Nodo, sólo se necesitan saber: la posición inicial y la posición objetivo, expresadas en ángulos de dirección (Yaw).

El Nodo de Navegación Autónoma puede consultarse en el Apéndice **L**. Implementado en Python, es exactamente el mismo Nodo que el **control_completo_node.py** de la Sección **5.2**,

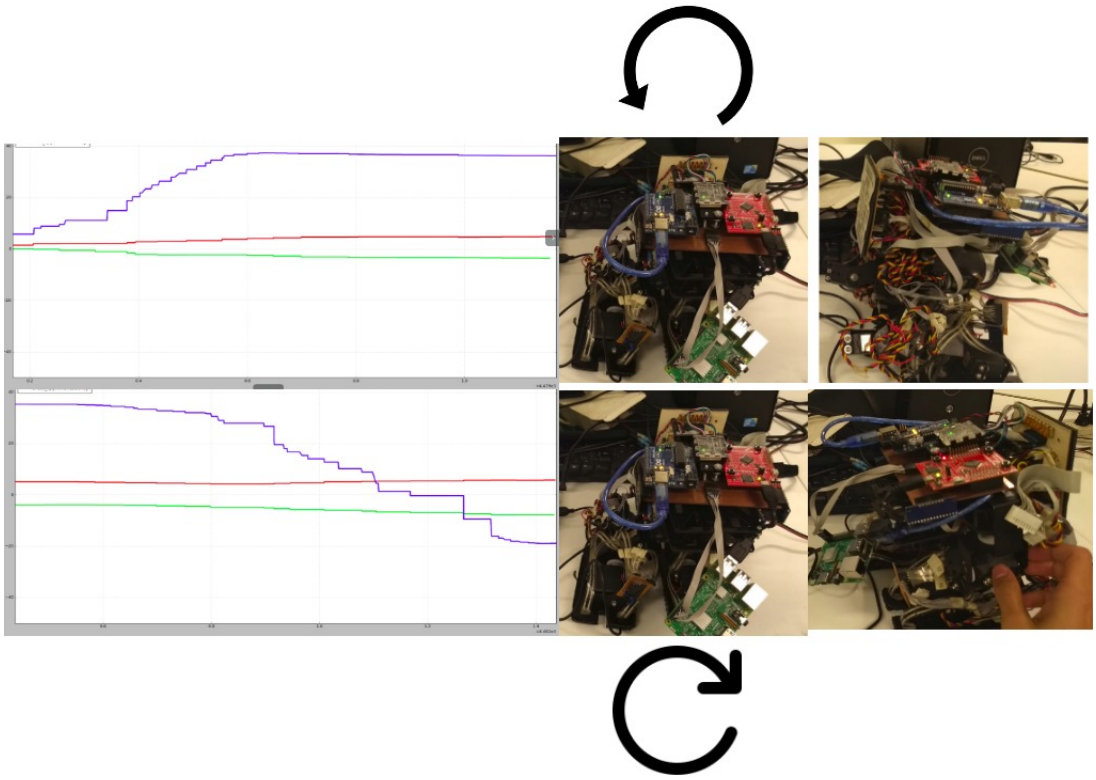

Figura 8.22: Prueba gráfica de variación de datos en Yaw proveniente de la MPU-6050 (**azul**)

con la diferencia de que la navegación autónoma realiza una modificación proporcional en los ángulos de ambas piernas en función del ángulo de dirección objetivo para realizar curvaturas en la caminata y así llegar a su destino.

El Paquete **Bipedo_CIA** contiene al Nodo **navegacion_autonoma_node.py** el cual se suscribe (por primera vez en todo el proyecto) a dos Tópicos: **/imu/rpy** de tipo **Vector3Stamped** con los datos de inclinación y **/yaw** de tipo **Float32** con el dato de dirección, publicando al Tópico **/servos_topic** de tipo **Float32MultiArray** los datos de PWM para los servomotores calculados en función de las trayectorias de marcha [5], la inclinación de la superficie [1] y el ángulo de dirección.

Finalmente, se manda con el **ROS MASTER**, teniendo a la Figura **8.23** como anexo al diagrama final del proyecto, siendo éste el de la Figura **8.24**.

Las Figuras **8.25** y **8.26** muestran los resultados de la navegación autónoma con ángulo inicial de **0°** y ángulos objetivo a **90°** y **180°** respectivamente, mostrando una caminata en curvas exitosa con el aprovechamiento del nuevo eje Yaw proporcionado por la MPU-6050.

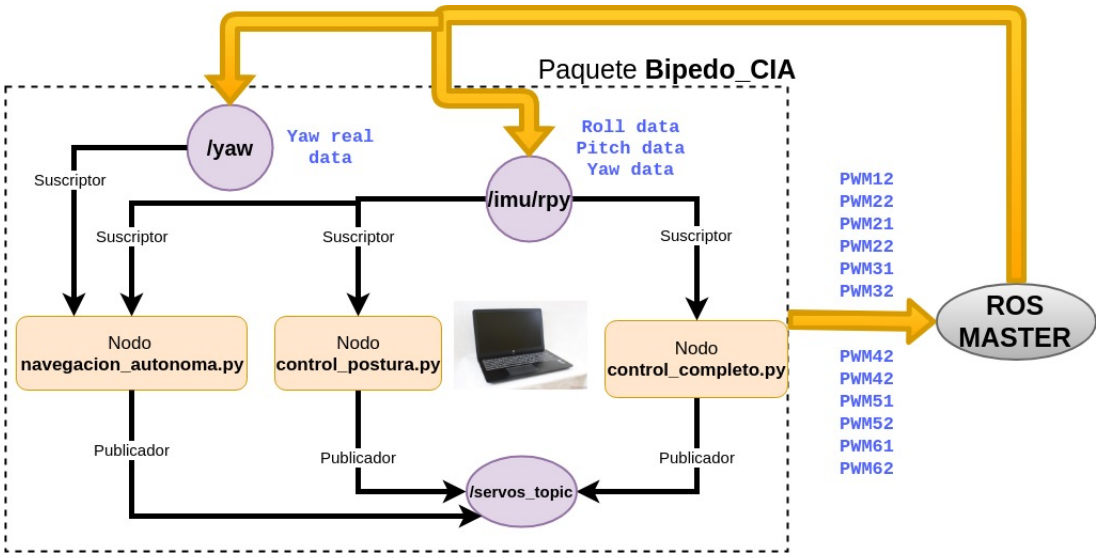

Figura 8.23: Nodo de Navegación Autónoma haciendo uso de Roll y Pitch de la UM7, y Yaw de la MPU-6050

8.5. Evaluación del desempeño del sistema

Para poder realizar una evaluación de ambos sistemas, fue necesario considerar referencias cualitativas (subjetivas) y cuantitativas (tangibles) para la tabulación, teniendo en cuenta al mejor valor como 5 y al peor como 0:

- **Controlabilidad de procesos**: Sin contar los Tópicos de graficación, en total hay **siete Tópicos** corriendo en el nuevo sistema con ROS, los cuales pueden controlarse mediante detención, visualización de sus propiedades, aprovechamiento de datos, etc. En cambio, en el sistema anterior era necesario *matar* el proceso por completo en la terminal única si se generaba algún error a lo largo de la operación y para ver los datos se tenían que imprimir forzosamente desde el código en Python. Este sistema no es totalmente incontrolable, pero no es un método que permita el óptimo desarrollo y observación de los procesos, dejándo al sistema anterior vulnerable ante fallas con una calificación aproximada de **tres**; en cambio, con ROS sí se logra esto, posicionándolo en la controlabilidad máxima con una calificación de **cinco**.

- **Estabilidad de procesos**: Analizando los datos de procesamiento de la Sección 8.2.3, se puede afirmar que el nuevo sistema con ROS aumenta considerablemente la estabilidad, mientras que el antiguo carece de ella. Tomando como punto de referencia el umbral de estabilidad sin fallas requerido (**0 fallas** en **63 segundos**), ROS logra el **100 %** de eficacia

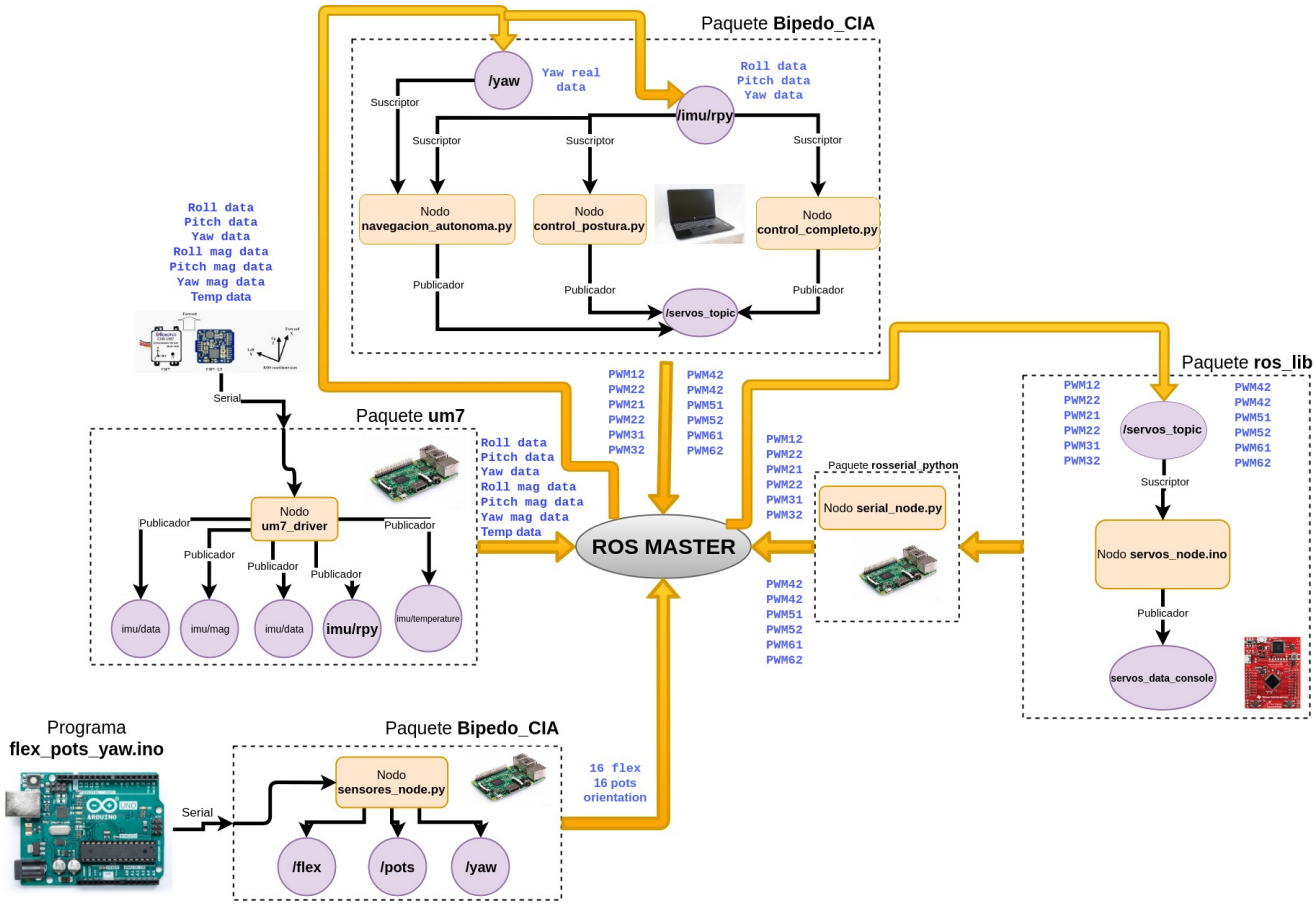

Figura 8.24: Diagrama de Nodos y Tópicos completo, incluyendo la navegación autónoma

en este objetivo, obteniendo la máxima calificación (**5**), mientras que el sistema anterior tuvo un promedio de **1 falla** cada **42 segundos**, dejándolo con **66 %** de eficacia, o **3.3** de calificación proporcionalmente.

- **Soporte y mantenimiento**: A pesar de necesitar más herramientas de instalación y conocimientos previos, ROS administra de mejor manera los procesos y facilita la programación de las leyes de control, ya que una vez designado el Nodo para la obtención de datos, por ejemplo los de Yaw, sólo es necesario generar el código para el aprovechamiento de esta variable, implicando un promedio entre **diez y quince líneas** elementales de código a modificarse/anexarse y tener un nuevo Nodo independiente para el robot. En cambio, el sistema anterior no tiene esta capacidad de soporte, ya que se necesita mandar llamar una función cada vez que se vaya a utilizar cualquier protocolo de comunicación, haciéndolo incompatible con otros procesos y perdiendo la esencia de modularización. Tomando como el caso ideal al nuevo sistema con calificación de **5**, se contempla que el antiguo sistema se queda con una

Figura 8.25: Prueba de navegación autónoma de **0°** a **90°** con giro derecho en **50 segundos**

calificación de **2**.

- **Velocidad de respuesta**: Con ROS se tiene una media de desfase de **223 milisegundos** (véase el por qué en la Sección 8.2). El sistema anterior presentaba un retraso cercano a **100 milisegundos**. Esto quiere decir que no existe demasiada diferencia entre ambos, pero sí una afectación en el tiempo de respuesta del nuevo sistema por el hecho de tener una herramienta de desarrollo más robusta. Tomando como referencia el sistema anterior como ideal (**5** de calificación) y como al peor sistema en donde el robot ya no puede caminar adecuadamente teniendo un retraso cercano a los **330 milisegundos**, proporcionalmente, el nuevo sistema con ROS obtiene una calificación de **2.3**.

- **Procesamiento de CPU y MEM**: Utilizando la herramienta Scout Realtime descrita en la Sección 8.2.3 para conocer los valores máximos y mínimos de las curvas de procesamiento, y teniendo en cuenta que a mayor consumo de CPU y MEM menor calificación (**100 %** como la peor), el sistema anterior recibe un **2.8** y un **4.1**, y el nuevo sistema con ROS un **3.9** y un **3.3** respectivamente.

Únicamente se evaluó la parte computacional, ya que el rediseño electrónico y el mantenimiento mecánico, a pesar de haberse mejorado, no es adecuado incluirlos directamente en la comparación puesto que forman parte de los requerimientos de hardware de la aplicación con ROS. Los números de la Tabla 8.4 indicados con * son las calificaciones propuestas con base en la experiencia del

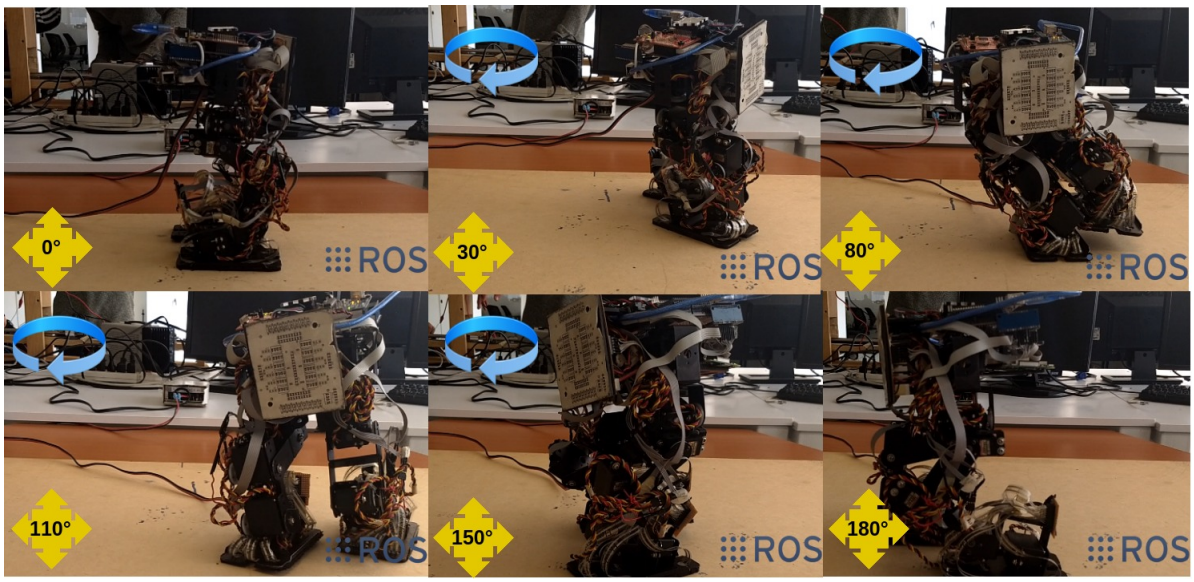

Figura 8.26: Prueba de navegación autónoma de **0°** a **180°** con giro derecho en **132 segundos**

desarrollo.

Tabla 8.4: Evaluación completa de ambos sistemas

Elemento a evaluar	**Evaluación sistema anterior [0-5]**	**Evaluación nuevo sistema con ROS [0-5]**	**Ponderación [0-10]**
Controlabilidad de procesos	3*	5	10*
Estabilidad de procesos	3	5	10*
Soporte y mantenimiento	2*	5*	10*
Velocidad de respuesta	5	2.3	9*
Procesamiento CPU	2.8	3.9	7*
Uso de MEM	4.1	3.3	6*
CALIFICACIÓN FINAL	**3.3**	**4.2**	

Capítulo 9

Conclusiones y trabajo a futuro

9.1. Conclusiones

Se pudo comprobar que, a pesar de que ROS incrementa ligeramente el retraso en la respuesta, **no repercute de manera trascendental en la evaluación visual ni en la funcional del robot**. A pesar de requerir mayor consumo de memoria, necesitar más conocimientos básicos en el manejo del software, o incluso el tiempo de la instalación previa, se puede confirmar que **ROS sí es una excelente herramienta de desarrollo para robots** como se menciona en la ROS Wiki [3] particularmente hablando del robot bípedo Scout. Gracias a ROS, el Bípedo mejoró significativamente la estabilidad de sus procesos computacionales, además de **sentar la base de un soporte de desarrollo para futuros proyectos**.

Realizando el Nodo de Navegación Autónoma y aprovechando el Tópico del eje de dirección Yaw, **se pudo verificar el trabajo colaborativo de la MPU-6050 y la UM7 para obtener finalmente los tres grados de libertad RPY como solución al sistema de referencia anteriormente incompleto**.

El magnetómetro de la UM7 no funcionó correctamente, pero fue intrascendente, ya que se necesitó dirección relativa y no orientación absoulta.

Realizando rediseño electrónico y mantenimiento mecánico, **se logró reducir la tasa de errores por falsos contactos, cortos circuitos y perturbaciones ajenas a los objetivos del control**.

Se lograron unificar y modularizar por completo los procesos de control anterior-

mente desarrollados para la postura y marcha bípeda.

9.2. Discusión

Como se mencionó en la Sección **8.2.2**, se tiene la hipótesis de que un publicar y un suscriptor no pueden coexistir eficientemente en el mismo microcontrolador al mismo tiempo, específicamente hablando del Arduino-UNO. Después de varios segundos, además de perder datos como se mostró en la comapración de gráficas en la Sección **8.2.1**, aparece un error de **lost sync** (sincronización perdida) entre el dispositivo y la Raspberry. Hay diveras causas de este fenómeno; probablemente la velocidad de transmisión de datos, incorrecta declaración de variables, desconexión física, variación repentina en la corriente de alimentación de la CPU, etc., pero lo investigado en los foros de ROS indica que el error no es particular.

Aparentemente se sobreponen el publicador y suscriptor respecto al número de mensajes enviados, evitando que los Tópicos funcionen de manera continua.

Desde cualquier perspectiva que se vea, es irrefutable el hecho de que ROS es superior al funcionamiento del sistema antiguo. Numéricamente hablando, se sabe que incrementa la calificación de su desempeño tomando en cuenta los factores de desarrollo más importantes abordados en esta tesis (Sección **8.5**). Los pesos de importancia de la Tabla **8.4** se encuentran a discusión porque se plantearon de manera aproximada. Desde luego, no se usó arbitrariedad. Todos tienen como justificación el conocimiento adquirido y analizado desde mucho antes de este trabajo profesional. Además, el fenómeno presentado coincide con numerosas discusiones y experimentos documentados en el blog oficial de ROS [45].

Como último punto de discusión, la Sección **4.6** menciona la estrategia de solución tomada para simular de manera confiable los procesos alámbricos del robot. Esto es relativo, ya que depende del equipo utilizado. Sin embargo, exactamente por la razón anterior se propuso realizar las pruebas en tres equipos diferentes para asegurarse de que no hubiera demasiada variación en la metodología de configuración del robot y sus herramientas. **¿Recomendación?**, si se utiliza ROS en un proyecto con monitoreo remoto y no se tiene una imagen funcional con todas las herramientas instaladas, es mejor primero simular los algoritmos con pruebas de escritorio.

9.3. Trabajo a futuro

Un desarrollo pendiente es implementar las trayectorias circulares realizadas por Fernanda Merino [5] en su tesis *Planeación de Trayectorias de un Robot Bípedo con un Modelo Parametrizado Carro Mesa*, pero ahora con un control en lazo cerrado con el dato de dirección.

Ahora, además de proceder a unificar los controladores basados en el ZMP (Zero Moment Point) y el CoM (Centro de Masa) para aumentar la robustez del sistema ante perturbaciones y mejorar la calidad de la caminata en línea recta, se encuentra disponible la posibilidad de implementar un mejor controlador para marcha en curvaturas gracias al aprovechamiento de datos del eje Yaw.

Usando de referencia principal el diagrama de bloques de la tesis anterior [1] y complementándolo en la Figura 9.1, se muestra con mayor claridad lo que se puede lograr en poco tiempo si se utiliza adecuadamente la herramienta llamada ROS, la cual modulariza los procesos de control y obtención de datos de los sensores.

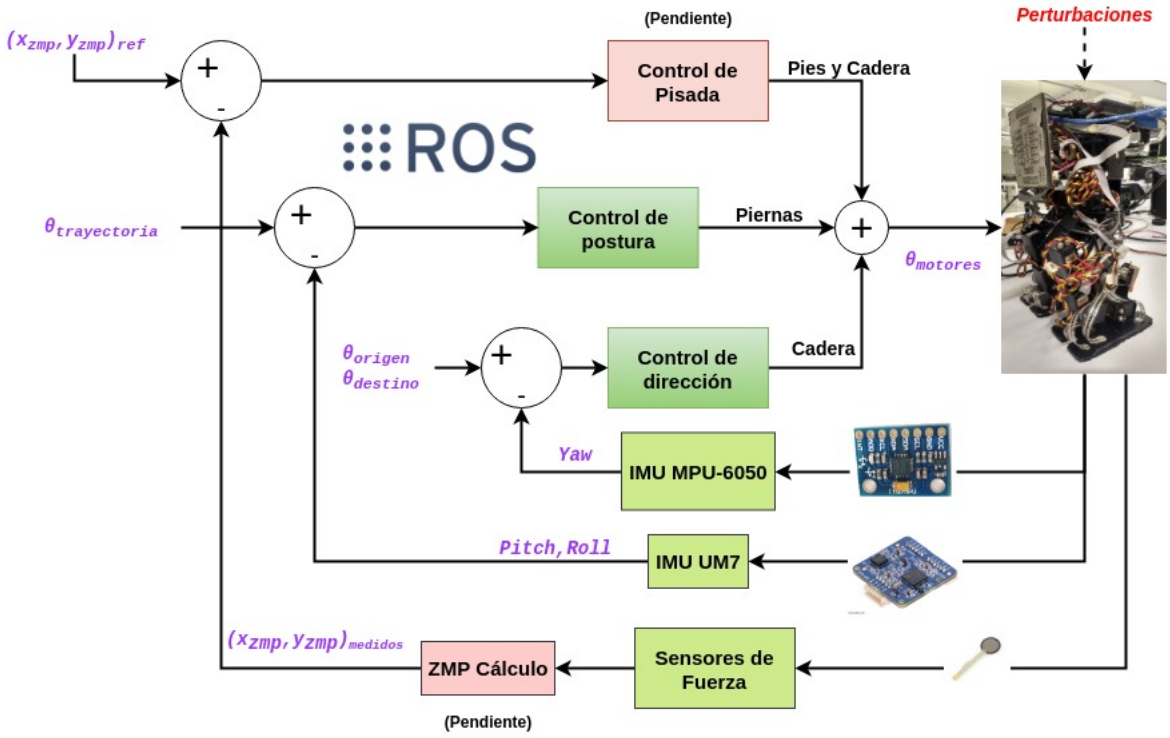

Figura 9.1: Nuevo diagrama de bloques propuesto con eje Yaw disponible y procesos modularizados con ROS

Una de las actuales investigaciones que se están implementando sobre el robot es llevada a cabo

por un estudiante de maestría de la facultad se llama *Control de Marcha de un Robot Bípedo con Operación Humana*, la cual desarrolla imitación robótica. Toda la plataforma de ROS ya cimentada ayudará a futuros proyectos como éste para migrar los programas sin mayor problema, ya que todo el respaldo se encuentra en los repositorios de github del Apéndice **M** utilizando la herramienta **gitclone**, así como en la computadora de escritorio del CIA.

Por último, queda pendiente una nueva evaluación mecánica, ya que se observó que, a lo largo de las pruebas, los servomotores han presentado un desgaste significativo, generando vibraciones no deseadas y desajustes en los ángulos iniciales.

Se piensa que los algoritmos del robot bípedo Scout tienen un gran potencial en el área de investigación y desarrollo, sobre todo por la era de crecimiento tecnológico que se está viviendo hoy en día. Todo el esfuerzo invertido, desde la obtención del equipo hasta las conclusiones actuales, han arrojado resultados satisfactorios para la formación de los futuros profesionistas de ésta institución, fomentando la investigación en nuestro país y generando conocimiento de distintas áreas de la ingeniería.

Bibliografía

[1] **SANCHEZ, Allen E.**, *Controladores difusos para postura y marcha de un robot bípedo de 12 GDL*. Tesis, Universidad Nacional Autónoma de México, Ciudad de México, 2017.

[2] **HERNÁNDEZ, Saddán**, *Renovación y mejoramiento de actuadores e instrumentación de un robot bípedo*. Tesis, Universidad Nacional Autónoma de México, Ciudad de México, 2016.

[3] **Open Source Robotics Foundation**. ROS.org: About ROS, accedido en 11/06/2018. http://www.ros.org/about-ros/, 2018.

[4] **Thingbits Electronics Pvt. Ltd.**. Lynxmotion Biped Robot Scout, accedido en 25/08/2018. https://www.thingbits.net/products/lynxmotion-biped-robot-scout-no-servos, 2018.

[5] **MORALES**, M. Fernanda, *Planeación de trayectorias de un robot Bípedo con un modelo parametrizado carro-mesa*. Tesis, Universidad Nacional Autónoma de México, Ciudad de México, 2016.

[6] **Random Nerd Tutorials**. Arduino vs Raspberry Pi vs BeagleBone vs PcDuino blog, accedido en 25/08/2018. https://randomnerdtutorials.com/arduino-vs-raspberry-pi-vs-beaglebone-vs-pcduino/, 2014.

[7] **MILOJICIC, Dean, KALOGERAKI, Vana, LUKO, Rajan, NAGARAJA, Kiran, PRUYNE, Jim, RICHARD, Bruno, RILLINS, Sami and XU, Zhichen**. Peer-to-Peer Computin. HP Laboratories Palo Alto : s.n., 2003.

[8] **Open Source Robotics Foundation**. ROS.org: Is ROS For Me?, accedido en 25/08/2018. http://www.ros.org/is-ros-for-me/, 2018.

[9] **JARBOLEYA**. Velneo: estabilidad en el software, accedido en 16/10/2018. https://velneo.es/estabilidad-en-el-software/, 2018.

[10] **TRIPOD Website**. Protocolo RS232, accedido en 11/06/2018,. http://rdedatos.tripod.com/rs232.htm, 2018.

[11] **Sparkfun**. Protocolo I^2C, accedido en 11/06/2018. https://learn.sparkfun.com/tutorials/i2c, 2011.

[12] Características de los sistemas de procedimientos remotos, accedido en 06/07/2018. http://bibing.us.es/proyectos/abreproy/12097/fichero/09-+Cap%C3%ADtulo+04-+Caracter%C3%ADsticas+de+los+sistemas+de+procedimientos+remotos.pdf, 2012.

[13] **BostonDynamics**. Página principal, accedido en 11/06/2018. https://www.bostondynamics.com/, 2018.

[14] **Institute for Robotics at Johannes Kepler University Linz**. ROS.org: BipedRobin, accedido en 12/06/2018. http://wiki.ros.org/Robots/BipedRobin, 2015.

[15] **Tecnología Informática website**. El sistema operativo, accedido en 13/06/2018. https://tecnologia-informatica.com/el-sistema-operativo/, 2017.

[16] **RIOJA, Ulises**, *Equilibrio dinámico de un robot bípedo utilizando el modelo simplificado carro-mesa*. Tesis, Universidad Nacional Autónoma de México, Ciudad de México, 2014.

[17] **ÁLVAREZ, Miguel A**, Desarrollador Web website. Qué es Python, accedido en 13/06/2018. https://desarrolloweb.com/articulos/1325.php, 2003.

[18] **BLANCHARD, David**. Introducción a C++, ¿qué es?, accedido en 13/06/2018. https://blanchardspace.wordpress.com/2013/05/06/introduccion-a-c-que-es/, 2017.

[19] **Texas Instruments**. Energia: Download Energia 18 aka Energia 1.6.10E18, accedido en 14/06/2018. http://energia.nu/download/, 2018.

[20] **Acutronic Robotics Company**. Introducción de ROS, accedido en 18/06/2018. https://erlerobotics.com/blog/ros-introduction-es/#concepts, 2018.

[21] **Open Source Robotics Foundation**. ROS.org: Ros/ Concepts, accedido en 14/06/2018. http://wiki.ros.org/ROS/Concepts, 2014.

[22] **Acutronic Robotics**. Introducción de ROS, accedido en 28/08/2018. https://erlerobotics.com/blog/ros-introduction-es/, 2018.

[23] **Cleapath Robotics**. ROS Tutorials: Intro to ROS, accedido en 14/06/2018. http://www.clearpathrobotics.com/assets/guides/ros, 2015.

[24] **Clearpath Robotics**. ROS Tutorials: ROS/TCPROS, accedido en 14/06/2018. http://wiki.ros.org/ROS/TCPROS, 2013.

[25] **Sublime HQ Pty Ltd**. Sublime Text: Download, accedido en 14/04/2018. https://www.sublimetext.com/3, 2018.

[26] **Tecnologia&Informatica**. Qué es el sistema operativo, accedido en 13/04/2018,. https://tecnologia-informatica.com/el-sistema-operativo/, 2017.

[27] **Mitch**. AskUbuntu: How do I install Terminator?, accedido en 13/04/2018. https://askubuntu.com/questions/829045/how-do-i-install-terminator, 2016.

[28] **Apnorton**. MATHEMATICS: Azimuth vs Yaw?, accedido en 04/05/2018. https://math.stackexchange.com/questions/296799/azimuth-vs-yaw, 2013.

[29] **Open Source Robotics Foundation**. ROS.org: ROS Kinetic Frame, accedido en 01/02/2018, http://wiki.ros.org/kinetic, 2018.

[30] **Texas Instruments**. Software development tools for your TI LaunchPad™ development kit, accedido en 14/06/2018. http://www.ti.com/tools-software/launchpads/software.html, 2018.

[31] **Open Source Robotics Foundation**. ROS.org: Arduino IDE Setup, accedido en 07/09/2018. http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup, 2018.

[32] **Texas Instruments**. Energia IDE, accedido en 14/06/2018. http://energia.nu/download/, 2018.

[33] **Pololu**. UM7-LT Orientation Sensor, accedido en 06/09/2018. https://www.pololu.com/product/2740, 2018.

[34] **Raspberry PI Foundation**. Raspberry Pi3 Model B, accedido en 06/09/2018. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/, 2018.

[35] **ROWBERG, Jeff**. Arduino: MPU-6050 Accelerometer + Gyro, accedido en 06/09/2018. https://playground.arduino.cc/Main/MPU-6050, 2011.

[36] **Clearpath Robotics**. ROS Tutorials: Launch Files, accedido en 14/06/2018. http://www.clearpathrobotics.com/assets/guides/ros/, 2015.

[37] **Autodesk**. EAGLE, PCB design made easy, accedido en 11/09/2018. https://www.autodesk.com/products/eagle/overview, 2018.

[38] **SADWOSKI, Mateusz**. Assign a static USB port on LINUX, accedido en 12/09/2018. https://msadowski.github.io/linux-static-port/, 2018.

[39] **Ubuntu MATE Team**. Ubuntu MATE 16.04 LTS, accedido en 15/09/2018. https://ubuntu-mate.org/download/, 2018.

[40] **Open Source Robotics Foundation**. ROS.org: Topics, accedido en 14/06/2018. http://wiki.ros.org/Topics, 2014.

[41] **Guru99**. Shell Scripting Tutorial for Linux/Unix Beginners, accedido en 19/09/2018. https://www.guru99.com/introduction-to-shell-scripting.html, 2018.

[42] **Raspberry PI Foundation**. RC.LOCAL, accedido en 20/09/2018. https://www.raspberrypi.org/documentation/linux/usage/rc-local.md, 2018.

[43] **Arduino**. ARDUINO UNO REV3, accedido en 31/08/2018. https://store.arduino.cc/usa/arduino-uno-rev3, 2018.

[44] **Open Source Robotics Foundation**. ROS.org: ROS Blog, accedido en 01/10/2018. http://www.ros.org/news/, 2018.

[45] **Open Source Robotics Foundation**. ROS.org: ROS Answers, accedido en 01/10/2018. https://answers.ros.org/questions/, 2018.

[46] **MATAMOROS, José M.**, *Análisis de extensibilidad, reestructuración y desempeño de software para robots móviles*. Tesis, Universidad Nacional Autónoma de México, Ciudad de México, 2013.

[47] **Texas Instruments**. Tiva C Series LaunchPads, accedido en 07/09/2018. http://processors.wiki.ti.com/index.php/Tiva_C_Series_LaunchPads, 2018.

[48] **Open Source Robotics Foundation**. ROS.org: um7, accedido en 14/06/2018. http://wiki.ros.org/um7, 2017.

[49] **Future Technology Devices International Ltd**. FTDI chip: usb-ttl serial cables, accedido en 14/06/2018. |http://www.ftdichip.com/Products/Cables/USBTTLSerial.htm, 2017.

[50] **Open Source Robotics Foundation**. ROS.org: rosserial-tivac, accedido en 14/06/2018. http://wiki.ros.org/rosserial/tivac/Tutorials/Energia, 2017.

[51] **SÁNCHEZ, Allen**. Youtube: Proyecto PAPIIT Robot Bípedo Scout 12 GDL, accedido en 04/11/2018. https://www.youtube.com/watch?v=TPskJykRhyY, 2017.

[52] **POSADAS, Antonio**, Determinación de errores y tratamiento de datos. Facultad de Ciencias Experimentales - Universidad de Almería, Departamento de Física Aplicada, Almería - España, 2017.

[53] **CAROLLO, M. Carmen**, Regresión lineal simple. Universidad de Santiago Compostella, Departamento de Estadística e Investigación Operativa, Santiago de Compostella - España, 2011-2012.

[54] **COLLINS, Ben**. How to make a Histogram in Google Sheets and overlay a Normal Distribution Curve, accedido en 10/01/2019. https://www.benlcollins.com/spreadsheets/histogram-in-google-sheets/, 2016.

[55] **Scout**. Scout Realtime: Top for the modern developer, accedido en 06/11/2018. https://scoutapp.github.io/scout_realtime/, 2018.

[56] **KENRICK, Mathew**. Quora: Is RAM size or processor speed more important?, accedido en 17/11/2018. https://www.quora.com/Is-RAM-size-or-processor-speed-more-important, 2018.

[57] **AZKUNE, Gorka**. Quantic Tells: Autonomus Navigation, accedido en 26/11/2018. https://cuentos-cuanticos.com/2011/11/12/navegacion-autonoma/, 2011.

[58] **Open Source Robotics Foundation**. ROS.org: rqt-plot, accedido en 07/10/2018. http://wiki.ros.org/rqt_plot, 2018.

[59] **JEGLALE, Jerome**. Sudo without password on Ubuntu, accedido en 15/07/2018. http://jeromejaglale.com/doc/unix/ubuntu_sudo_without_password, 2008.

Apéndice A

control_postura_node.py (laptop o de escritorio)

```
#!/usr/bin/env python

# ---------- DEPENDENCIAS Y BIBLIOTECAS ----------
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import time
from time import time as Time_c
import random
import math

import rospy
from std_msgs.msg import Int32MultiArray
from geometry_msgs.msg import Vector3Stamped
from geometry_msgs.msg import Pose

dim_captura = 820
datos_recabados = np.zeros((dim_captura,33))

eP_pitch = np.arange(-90, 90,0.1)

eP_pitch_ENG = fuzz.trimf(eP_pitch, [-90,-90,-15])#error negativo grande
eP_pitch_ENP = fuzz.trimf(eP_pitch,[-60,-15,0])#error negativo pequeno
eP_pitch_ECC = fuzz.trapmf(eP_pitch,[-15,-5,5,15])#error cercano cero
eP_pitch_EPP = fuzz.trimf(eP_pitch,[0,15,60])#error positivo pequeno
```

```
eP_pitch_EPG = fuzz.trimf(eP_pitch, [15,90,90])#error positivo grande

eI_pitch = np.arange(-200, 200,0.1)

eI_pitch_ENG = fuzz.trimf(eI_pitch, [-200,-200,-50])#error negativo grande
eI_pitch_ENP = fuzz.trimf(eI_pitch,[-100,-50,0])#error negativo pequeno
eI_pitch_ECC = fuzz.trimf(eI_pitch,[-50,0,50])#error cercano cero
eI_pitch_EPP = fuzz.trimf(eI_pitch,[0,50,100])#error positivo pequeno
eI_pitch_EPG = fuzz.trimf(eI_pitch, [50,200,200])#error positivo grande

eD_pitch = np.arange(-300, 300,0.1)

eD_pitch_ENG = fuzz.trimf(eD_pitch, [-300,-300,-50])#error negativo grande
eD_pitch_ENP = fuzz.trimf(eD_pitch,[-100,-50,0])#error negativo pequeno
eD_pitch_ECC = fuzz.trapmf(eD_pitch,[-50,-5,5,50])#error cercano cero
eD_pitch_EPP = fuzz.trimf(eD_pitch,[0,50,100])#error positivo pequeno
eD_pitch_EPG = fuzz.trimf(eD_pitch, [50,300,300])#error positivo grande

eP_roll = np.arange(-90, 90,0.1)

eP_roll_ENG = fuzz.trimf(eP_roll, [-90,-90,-15])#error negativo grande
eP_roll_ENP = fuzz.trimf(eP_roll,[-60,-25,0])#error negativo pequeno
eP_roll_ECC = fuzz.trapmf(eP_roll,[-25,-5,5,25])#error cercano cero
eP_roll_EPP = fuzz.trimf(eP_roll,[0,25,60])#error positivo pequeno
eP_roll_EPG = fuzz.trimf(eP_roll, [15,90,90])#error positivo grande

KP = np.arange(0,1.5,0.1)#angulo inicial robot: 115

KP_mf1 = fuzz.trapmf(KP, [0, 0,0.2,0.4])#angulo cercano cero
KP_mf2 = fuzz.trimf(KP, [0.2, 0.4, 0.6])#angulo pequeno cero
KP_mf3 = fuzz.trimf(KP,[0.4,0.6,0.8])#angulo mitad
KP_mf4 = fuzz.trimf(KP, [0.6, 0.9, 1.1])#angulo pequeno mitad
KP_mf5 = fuzz.trapmf(KP, [0.9, 1.1, 1.5,1.5])#angulo casi final


KI = np.arange(0,1.5,0.1)#angulo inicial robot: 65

KI_mf1 = fuzz.trapmf(KI, [0, 0,0.2,0.4])#angulo cercano cero
KI_mf2 = fuzz.trimf(KI, [0.2, 0.4, 0.6])#angulo pequeno cero
KI_mf3 = fuzz.trimf(KI,[0.4,0.6,0.8])#angulo mitad
KI_mf4 = fuzz.trimf(KI, [0.6, 0.9, 1.1])#angulo pequeno mitad
KI_mf5 = fuzz.trapmf(KI, [0.9, 1.1, 1.5,1.5])#angulo casi final

KD = np.arange(0,0.5,0.1)#angulo inicial robot: 140

```

```
KD_mf1 = fuzz.trimf(KD,[ 0,0,0.1])#angulo cercano cero
KD_mf2 = fuzz.trimf(KD, [0, 0.1, 0.2])#angulo pequeno cero
KD_mf3 = fuzz.trimf(KD,[0.1,0.2,0.3])#angulo mitad
KD_mf4 = fuzz.trimf(KD, [0.2, 0.3, 0.4])#angulo pequeno mitad
KD_mf5 = fuzz.trapmf(KD, [0.3, 0.4, 0.5,0.5])#angulo casi final

KP_roll = np.arange(0,1.5,0.1)#angulo inicial robot: 115

KP_roll_mf1 = fuzz.trapmf(KP_roll, [0, 0,0.1,0.2])#angulo cercano cero
KP_roll_mf2 = fuzz.trimf(KP_roll, [0.1, 0.2, 0.4])#angulo pequeno cero
KP_roll_mf3 = fuzz.trimf(KP_roll,[0.2,0.4,0.8])#angulo mitad
KP_roll_mf4 = fuzz.trimf(KP_roll, [0.6, 0.8, 1.1])#angulo pequeno mitad
KP_roll_mf5 = fuzz.trapmf(KP_roll, [0.8, 1.1, 1.5,1.5])#angulo casi final

KP_roll_c = np.arange(0,2.5,0.1)#angulo inicial robot: 115

KP_roll_c_mf1 = fuzz.trapmf(KP_roll_c, [0, 0,0.4,0.8])#angulo cercano cero
KP_roll_c_mf2 = fuzz.trimf(KP_roll_c, [0.4, 0.8, 1.2])#angulo pequeno cero
KP_roll_c_mf3 = fuzz.trimf(KP_roll_c,[0.8,1.2,1.6])#angulo mitad
KP_roll_c_mf4 = fuzz.trimf(KP_roll_c, [1.2, 1.6, 2.0])#angulo pequeno mitad
KP_roll_c_mf5 = fuzz.trapmf(KP_roll_c, [1.6, 2.0, 2.5,2.5])#angulo casi final

Ang_RodillaD = np.arange(0,35,1)#angulo inicial robot: 140

Ang_RodillaD_mf1 = fuzz.trapmf(Ang_RodillaD,[0,0,10,15])#angulo cercano cero
Ang_RodillaD_mf2 = fuzz.trimf(Ang_RodillaD,[10, 15,20])#angulo pequeno cero
Ang_RodillaD_mf3 = fuzz.trimf(Ang_RodillaD,[15,20,25])#angulo mitad
Ang_RodillaD_mf4 = fuzz.trimf(Ang_RodillaD,[20,25,30])#angulo pequeno mitad
Ang_RodillaD_mf5 = fuzz.trapmf(Ang_RodillaD,[25,30,35, 35])#angulo casi final

Ang_RodillaI = np.arange(0,35,1)#angulo inicial robot: 140

Ang_RodillaI_mf1 = fuzz.trapmf(Ang_RodillaI,[0,0,10,15])#angulo cercano cero
Ang_RodillaI_mf2 = fuzz.trimf(Ang_RodillaI,[10, 15,20])#angulo pequeno cero
Ang_RodillaI_mf3 = fuzz.trimf(Ang_RodillaI,[15,20,25])#angulo mitad
Ang_RodillaI_mf4 = fuzz.trimf(Ang_RodillaI,[20,25,30])#angulo pequeno mitad
Ang_RodillaI_mf5 = fuzz.trapmf(Ang_RodillaI,[25,30,35,35])#angulo casi final


def ErrorP_category(error_in):
    eP_pitch_ENG_cat = fuzz.interp_membership(eP_pitch,eP_pitch_ENG,error_in)
    eP_pitch_ENP_cat = fuzz.interp_membership(eP_pitch,eP_pitch_ENP,error_in)
    eP_pitch_ECC_cat = fuzz.interp_membership(eP_pitch,eP_pitch_ECC,error_in)
    eP_pitch_EPP_cat = fuzz.interp_membership(eP_pitch,eP_pitch_EPP,error_in)
    eP_pitch_EPG_cat = fuzz.interp_membership(eP_pitch,eP_pitch_EPG,error_in)
```

```
116     return dict(NG = eP_pitch_ENG_cat, NP = eP_pitch_ENP_cat, CE = eP_pitch_ECC_cat
        , PP = eP_pitch_EPP_cat, PG = eP_pitch_EPG_cat)
117
118 def ErrorI_category(error_in):
119     eI_pitch_ENG_cat = fuzz.interp_membership(eI_pitch,eI_pitch_ENG,error_in)
120     eI_pitch_ENP_cat = fuzz.interp_membership(eI_pitch,eI_pitch_ENP,error_in)
121     eI_pitch_ECC_cat = fuzz.interp_membership(eI_pitch,eI_pitch_ECC,error_in)
122     eI_pitch_EPP_cat = fuzz.interp_membership(eI_pitch,eI_pitch_EPP,error_in)
123     eI_pitch_EPG_cat = fuzz.interp_membership(eI_pitch,eI_pitch_EPG,error_in)
124     return dict(NG = eI_pitch_ENG_cat, NP = eI_pitch_ENP_cat, CE = eI_pitch_ECC_cat
        , PP = eI_pitch_EPP_cat, PG = eI_pitch_EPG_cat)
125
126 def ErrorD_category(error_in):
127     eD_pitch_ENG_cat = fuzz.interp_membership(eD_pitch,eD_pitch_ENG,error_in)
128     eD_pitch_ENP_cat = fuzz.interp_membership(eD_pitch,eD_pitch_ENP,error_in)
129     eD_pitch_ECC_cat = fuzz.interp_membership(eD_pitch,eD_pitch_ECC,error_in)
130     eD_pitch_EPP_cat = fuzz.interp_membership(eD_pitch,eD_pitch_EPP,error_in)
131     eD_pitch_EPG_cat = fuzz.interp_membership(eD_pitch,eD_pitch_EPG,error_in)
132     return dict(NG = eD_pitch_ENG_cat, NP = eD_pitch_ENP_cat, CE = eD_pitch_ECC_cat
        , PP = eD_pitch_EPP_cat, PG = eD_pitch_EPG_cat)
133
134 def ErrorP_roll_category(error_in):
135     eP_roll_ENG_cat = fuzz.interp_membership(eP_roll,eP_roll_ENG,error_in)
136     eP_roll_ENP_cat = fuzz.interp_membership(eP_roll,eP_roll_ENP,error_in)
137     eP_roll_ECC_cat = fuzz.interp_membership(eP_roll,eP_roll_ECC,error_in)
138     eP_roll_EPP_cat = fuzz.interp_membership(eP_roll,eP_roll_EPP,error_in)
139     eP_roll_EPG_cat = fuzz.interp_membership(eP_roll,eP_roll_EPG,error_in)
140     return dict(NG = eP_roll_ENG_cat, NP = eP_roll_ENP_cat, CE = eP_roll_ECC_cat,
        PP = eP_roll_EPP_cat, PG = eP_roll_EPG_cat)
141
142
143 def Limites(angulo):
144     if angulo > 180:
145         angulo = 180
146     if angulo < 0:
147         angulo = 0
148     return angulo
149
150 # ---------------------------- VARIABLES GLOBALES ----------------------------
151 PWM = [0,0,0,0,0,0,0,0,0,0,0,0] # 12 Servomotores
152 RPY = [0.0,0.0,0.0]      # Arreglo donde se leen Roll, Pitch & Yaw
153
154 ahora_t = 0
155 ultimo_t = 0
156 dt = 0
```

```
157 error_sum = 0
158 contador = 0
159 tiempo = 0
160 suma_tiempo = 0
161 filter_rate = 0
162 error_roll = 0
163 error_pitch = 0
164 Kp = 0
165 Ki = 0
166 i = 0
167 U_21 = 0
168 U_22 = 0
169 U_31 = 0
170 U_32 = 0
171 U_41 = 0
172 U_42 = 0
173 U_51 = 0
174 U_52 = 0
175 U_61 = 0
176 U_62 = 0
177
178 # ================== CALLBACK de ROS ============================
179 def CallBack(RPY_data):
180     # Uso de variables globales
181     global ahora_t, ultimo_t, error_roll, error_pitch, filter_rate
182     global dt, error_sum, contador, tiempo, suma_tiempo, i
183     global U_21, U_22, U_31, U_32, U_41, U_42, U_51, U_52, U_61, U_62
184
185     ahora_t = Time_c() # Para comenzar a medir tiempo
186     # Almacenamiento de datos de la IMU
187     RPY[0] = RPY_data.vector.x*57.2958
188     RPY[1] = RPY_data.vector.y*57.2958
189     RPY[2] = RPY_data.vector.z*57.2958
190
191     # Calculo de los errores
192     error_roll = 0 - RPY[0]
193     error_pitch = 0 - RPY[1]
194     error_rate = 0
195     filter_rate = filter_rate + 0.1*(error_rate - filter_rate)
196     ultimo_t = Time_c() # Para terminar de medir tiempo
197     dt = ultimo_t - ahora_t # Diferencial de tiempo
198     suma_tiempo  += dt
199     error_sum += (error_pitch*dt)
200
201     #categorias errores pitch
```

```
202     eP_rule = ErrorP_category(error_pitch)
203     eI_rule = ErrorI_category(error_sum)
204     eD_rule = ErrorD_category(filter_rate)
205 
206     #categorias error roll
207     eP_roll_rule = ErrorP_roll_category(error_roll)
208 
209     #ERROR PITCH
210     regla1_KP = np.fmax(eP_rule['NG'],eD_rule['NG'])
211     regla2_KP = np.fmax(eP_rule['NP'],eD_rule['NP'])
212     regla3_KP = np.fmax(eP_rule['CE'],eD_rule['CE'])
213     regla4_KP = np.fmax(eP_rule['PP'],eD_rule['PP'])
214     regla5_KP = np.fmax(eP_rule['PG'],eD_rule['PG'])
215 
216     #ERROR SUMA PITCH
217     regla1_KI = np.fmax(eP_rule['NG'],eI_rule['NG'])
218     regla2_KI = np.fmax(eP_rule['NP'],eI_rule['NP'])
219     regla3_KI = np.fmax(eP_rule['CE'],eI_rule['CE'])
220     regla4_KI = np.fmax(eP_rule['PP'],eI_rule['PP'])
221     regla5_KI = np.fmax(eP_rule['PG'],eI_rule['PG'])
222 
223     #CAMBIO ERROR  PITCH
224     regla1_KD = np.fmax(eP_rule['NG'],eD_rule['NG'])
225     regla2_KD = eP_rule['NP']
226     regla3_KD = eP_rule['CE']
227     regla4_KD = eP_rule['PP']
228     regla5_KD = np.fmax(eP_rule['PG'],eD_rule['PG'])
229 
230     #ERROR ROLL
231     regla1_KP_roll = eP_roll_rule['NG']
232     regla2_KP_roll = eP_roll_rule['NP']
233     regla3_KP_roll = eP_roll_rule['CE']
234     regla4_KP_roll = eP_roll_rule['PP']
235     regla5_KP_roll = eP_roll_rule['PG']
236 
237     #ERROR ROLL cadera
238     regla1_KP_roll_c = eP_roll_rule['NG']
239     regla2_KP_roll_c = eP_roll_rule['NP']
240     regla3_KP_roll_c = eP_roll_rule['CE']
241     regla4_KP_roll_c = eP_roll_rule['PP']
242     regla5_KP_roll_c = eP_roll_rule['PG']
243 
244     #ANGULO RODILLA
245     regla1_ang_rodillaD = eP_roll_rule['NG']
246     regla2_ang_rodillaD = eP_roll_rule['NP']
```

```
247     regla3_ang_rodillaD = eP_roll_rule['CE']
248     regla4_ang_rodillaD = eP_roll_rule['PP']
249     regla5_ang_rodillaD = eP_roll_rule['PG']
250 
251     #ANGULO RODILLA
252     regla1_ang_rodillaI = eP_roll_rule['NG']
253     regla2_ang_rodillaI = eP_roll_rule['NP']
254     regla3_ang_rodillaI = eP_roll_rule['CE']
255     regla4_ang_rodillaI = eP_roll_rule['PP']
256     regla5_ang_rodillaI = eP_roll_rule['PG']
257 
258     #Implementacion de reglas KP,KI,KD
259     regla1_KP_act = np.fmin(regla1_KP,KP_mf5)#si ENG y  entonces KP mf5 nivel alto
260     regla2_KP_act = np.fmin(regla2_KP,KP_mf3)#si ENP y entonces KP mf2, nivel bajo
261     regla3_KP_act = np.fmin(regla3_KP,KP_mf1)#si ECC y entonces KP mf3
262     regla4_KP_act = np.fmin(regla4_KP,KP_mf3)#si EPP y entonces KP mf4
263     regla5_KP_act = np.fmin(regla5_KP,KP_mf5)#si EPG y entonces KP mf5
264 
265     regla1_KI_act = np.fmin(regla1_KI,KI_mf1)#si ENG y ENG entonces KI mf1
266     regla2_KI_act = np.fmin(regla2_KI,KI_mf2)#si ENP y entonces KI mf2
267     regla3_KI_act = np.fmin(regla3_KI,KI_mf4)#si ECC y entonces KI mf3
268     regla4_KI_act = np.fmin(regla4_KI,KI_mf2)#si EPP y entonces KI mf4
269     regla5_KI_act = np.fmin(regla5_KI,KI_mf1)#si EPG y entonces KI mf5
270 
271     regla1_KD_act = np.fmin(regla1_KD,KD_mf5)#si ENG y ENG entonces KD mf1
272     regla2_KD_act = np.fmin(regla2_KD,KD_mf2)#si ENP y entonces KD mf2
273     regla3_KD_act = np.fmin(regla3_KD,KD_mf1)#si ECC y entonces KD mf3
274     regla4_KD_act = np.fmin(regla4_KD,KD_mf2)#si EPP y entonces KD mf4
275     regla5_KD_act = np.fmin(regla5_KD,KD_mf5)#si EPG y entonces KD mf5
276 
277     regla1_KP_roll_act = np.fmin(regla1_KP_roll,KP_roll_mf5)#si ENG y  entonces KP
    mf5 nivel alto
278     regla2_KP_roll_act = np.fmin(regla2_KP_roll,KP_roll_mf3)#si ENP y entonces KP
    mf2, nivel bajo
279     regla3_KP_roll_act = np.fmin(regla3_KP_roll,KP_roll_mf1)#si ECC y entonces KP
    mf3
280     regla4_KP_roll_act = np.fmin(regla4_KP_roll,KP_roll_mf3)#si EPP y entonces KP
    mf4
281     regla5_KP_roll_act = np.fmin(regla5_KP_roll,KP_roll_mf5)#si EPG y entonces KP
    mf5
282 
283     regla1_KP_roll_c_act = np.fmin(regla1_KP_roll_c,KP_roll_c_mf5)#si ENG y
    entonces KP mf5 nivel alto
284     regla2_KP_roll_c_act = np.fmin(regla2_KP_roll_c,KP_roll_c_mf2)#si ENP y
    entonces KP mf2, nivel bajo
```

```
285     regla3_KP_roll_c_act = np.fmin(regla3_KP_roll_c,KP_roll_c_mf1)#si ECC y
    entonces KP mf3
286     regla4_KP_roll_c_act = np.fmin(regla4_KP_roll_c,KP_roll_c_mf2)#si EPP y
    entonces KP mf4
287     regla5_KP_roll_c_act = np.fmin(regla5_KP_roll_c,KP_roll_c_mf5)#si EPG y
    entonces KP mf5
288
289     regla1_ang_rodillaD_act = np.fmin(regla1_ang_rodillaD,Ang_RodillaD_mf5)#si ENG
    y ENG entonces KD mf1
290     regla2_ang_rodillaD_act = np.fmin(regla2_ang_rodillaD,Ang_RodillaD_mf4)#si ENP
    y entonces KD mf2
291     regla3_ang_rodillaD_act = np.fmin(regla3_ang_rodillaD,Ang_RodillaD_mf2)#si ECC
    y entonces KD mf3
292     regla4_ang_rodillaD_act = np.fmin(regla4_ang_rodillaD,Ang_RodillaD_mf4)#si EPP
    y entonces KD mf4
293     regla5_ang_rodillaD_act = np.fmin(regla5_ang_rodillaD,Ang_RodillaD_mf5)#si EPG
    y entonces KD mf5
294
295     regla1_ang_rodillaI_act = np.fmin(regla1_ang_rodillaI,Ang_RodillaI_mf5)#si ENG
    y ENG entonces KD mf1
296     regla2_ang_rodillaI_act = np.fmin(regla2_ang_rodillaI,Ang_RodillaI_mf4)#si ENP
    y entonces KD mf2
297     regla3_ang_rodillaI_act = np.fmin(regla3_ang_rodillaI,Ang_RodillaI_mf2)#si ECC
    y entonces KD mf3
298     regla4_ang_rodillaI_act = np.fmin(regla4_ang_rodillaI,Ang_RodillaI_mf4)#si EPP
    y entonces KD mf4
299     regla5_ang_rodillaI_act = np.fmin(regla5_ang_rodillaI,Ang_RodillaI_mf5)#si EPG
    y entonces KD mf5
300
301     SumaMembresiasKP = np.fmax(regla1_KP_act,np.fmax(regla2_KP_act,np.fmax(
    regla3_KP_act,np.fmax(regla4_KP_act,regla5_KP_act))))
302     SumaMembresiasKI =  np.fmax(regla1_KI_act,np.fmax(regla2_KI_act,np.fmax(
    regla3_KI_act,np.fmax(regla4_KI_act,regla5_KI_act))))
303     SumaMembresiasKD = np.fmax(regla1_KD_act,np.fmax(regla2_KD_act,np.fmax(
    regla3_KD_act,np.fmax(regla4_KD_act,regla5_KD_act))))
304
305     SumaMembresiasKP_roll = np.fmax(regla1_KP_roll_act,np.fmax(regla2_KP_roll_act,
    np.fmax(regla3_KP_roll_act,np.fmax(regla4_KP_roll_act,regla5_KP_roll_act))))
306     SumaMembresiasKP_roll_c = np.fmax(regla1_KP_roll_c_act,np.fmax(
    regla2_KP_roll_c_act,np.fmax(regla3_KP_roll_c_act,np.fmax(regla4_KP_roll_c_act,
    regla5_KP_roll_c_act))))
307
308     SumaMembresias_a_D = np.fmax(regla1_ang_rodillaD_act,np.fmax(
    regla2_ang_rodillaD_act,np.fmax(regla3_ang_rodillaD_act,np.fmax(
    regla4_ang_rodillaD_act,regla5_ang_rodillaD_act))))
```

```
309     SumaMembresias_a_I = np.fmax(regla1_ang_rodillaI_act ,np.fmax(
        regla2_ang_rodillaI_act ,np.fmax(regla3_ang_rodillaI_act ,np.fmax(
        regla4_ang_rodillaI_act ,regla5_ang_rodillaI_act))))
310
311     #defuzzification , resultado:
312     Kp = fuzz.centroid(KP, SumaMembresiasKP)
313     Ki = fuzz.centroid(KI, SumaMembresiasKI)
314     Kd = fuzz.centroid(KD, SumaMembresiasKD)
315
316     Kp_roll = fuzz.centroid(KP_roll, SumaMembresiasKP_roll)
317     Kp_roll_c = fuzz.centroid(KP_roll_c, SumaMembresiasKP_roll_c)
318
319     a_D = fuzz.centroid(Ang_RodillaD, SumaMembresias_a_D)
320     a_I = fuzz.centroid(Ang_RodillaI, SumaMembresias_a_I)
321
322     U_21 = 90 + Kp_roll_c*error_roll
323     U_22 = 90 + Kp_roll_c*error_roll
324     #Originales Allen:
325     '''U_31 = 160 - 0.2479*error_pitch - 0.804*error_sum - 0.056*error_rate - a_D
326     U_32 = 35 + 0.2479*error_pitch + 0.804*error_sum + 0.056*error_rate +  a_I
327     U_41 = 65 - 0.2479*error_pitch - 0.804*error_sum - 0.056*error_rate - a_D
328     U_42 = 120 + 0.2479*error_pitch + 0.804*error_sum + 0.056*error_rate + a_I
329     U_51 = 120 - 0.2479*error_pitch - 0.804*error_sum - 0.056*error_rate - a_D
330     U_52 = 65 + 0.2479*error_pitch + 0.804*error_sum + 0.056*error_rate + a_I'''
331     U_31 = 160 - 0.8479*error_pitch - 1.004*error_sum - 0.756*error_rate - a_D
332     U_32 = 35 + 0.8479*error_pitch + 1.004*error_sum + 0.756*error_rate +  a_I
333     U_41 = 65 - 0.8479*error_pitch - 1.004*error_sum - 0.756*error_rate - a_D
334     U_42 = 120 + 0.8479*error_pitch + 1.004*error_sum + 0.756*error_rate + a_I
335     U_51 = 120 - 0.8479*error_pitch - 1.004*error_sum - 0.756*error_rate - a_D
336     U_52 = 65 + 0.8479*error_pitch + 1.004*error_sum + 0.756*error_rate + a_I
337     U_61 = 90 + Kp_roll*error_roll
338     U_62 = 90 + Kp_roll*error_roll
339
340     servo11 = 90
341     servo12 = 90
342     servo21 = Limites(int(U_21))
343     servo22 = Limites(int(U_22))
344     servo31 = Limites(int(U_31))#cadera PID
345     servo32 = Limites(int(U_32))#cadera PID
346     servo41 = Limites(int(U_41))#Rodilla
347     servo42 = Limites(int(U_42))#Rodilla
348     servo51 = Limites(int(U_51))#Tobillo PID
349     servo52 = Limites(int(U_52))#Tobillo PID
350     servo61 = Limites(int(U_61))
351     servo62 = Limites(int(U_62))
```

```
352
353     PWM[0] = servo11              # servo11
354     PWM[1] = servo12              # servo12
355     PWM[2] = servo21              # servo21
356     PWM[3] = servo22              # servo22
357     PWM[4] = servo31#cadera PID   # servo31
358     PWM[5] = servo32#cadera PID   # servo32
359     PWM[6] = servo41#Rodilla      # servo41
360     PWM[7] = servo42#Rodilla      # servo42
361     PWM[8] = servo51#Tobillo PID # servo51
362     PWM[9] = servo52#Tobillo PID # servo52
363     PWM[10] = servo61             # servo61
364     PWM[11] = servo62             # servo62
365
366 # ====================== Publicador y Subscriptor ======================
367 def RPY_listener_and_PWM_talker():
368
369     # Nombre del nodo, false para que no despliegue numero aleatorio
370     rospy.init_node('control_postura_node', anonymous=False)
371     print("Inicializando control_postura_node...")
372
373     # Definicion del publicador
374     pub = rospy.Publisher('/servos_topic', Int32MultiArray, queue_size=10)
375     pub_plot = rospy.Publisher('/servo_plot_compu', Pose, queue_size=10)
376
377     # Definicion del subscriptor
378     rospy.Subscriber('/imu/rpy', Vector3Stamped, CallBack)
379     #rospy.spin() <-------- este es remplazado por while not rospy.is_shutdown()
380
381     # Velocidad del programa
382     rate = rospy.Rate(500) # (10) = 10[Hz]
383
384     # Arreglo de 12 servos donde se almacenaran los datos a publicar
385     angulos = Int32MultiArray()
386     angulo_plot = Pose()
387
388     # Angulos iniciales
389     angulos.data = [90,90,90,87,141,57,60,120,120,60,87,87]
390     angulo_plot.position.x = 0
391
392     # ============== WHILE LOOP ============================
393     while not rospy.is_shutdown():  # Confirma que todo esta bien
394         start = time.time()
395         # Uso de variables globales
396         global PWM, error_roll, error_pitch, error_sum, filter_rate, Kp, Ki,
```

```
    datos_recabados
397
398         # Almacenamiento en arreglo del publicador
399         if RPY[0] == 0 and RPY[1] == 0:
400           angulos.data = [90,90,90,87,138,60,60,120,120,60,87,87];
401         else:
402             angulos.data[0] = PWM[0]
403             angulos.data[1] = PWM[1]
404             angulos.data[2] = PWM[2]
405             angulos.data[3] = PWM[3]
406             angulos.data[4] = PWM[4]
407             angulos.data[5] = PWM[5]
408             angulos.data[6] = PWM[6]
409             angulos.data[7] = PWM[7]
410             angulos.data[8] = PWM[8]
411             angulos.data[9] = PWM[9]
412             angulos.data[10] = PWM[10]
413             angulos.data[11] = PWM[11]
414
415             angulo_plot.position.x = angulos.data[10] # el servo de posicion 10 de
    graficacion
416
417         # Impresion de datos de calculo
418         #print "eR: ",error_roll,"eP: ", error_pitch," eI: ",error_sum," eD: ",
    filter_rate, " Kp: ",Kp, " Ki: ",Ki
419         #print "Roll: ", RPY[0], " Pitch: ", RPY[1]
420         # Guarda un archivo de texto en la misma ruta de los datos recabados
421         np.savetxt('dp_PID.txt',datos_recabados,fmt = '%10.5f', delimiter = '\t')
422
423         # Publicacion del mensaje
424         pub.publish(angulos)  #publicador de datos
425         #pub_plot.publish(angulo_plot) #publicador de graficacion
426
427         # Delay del programa definido en el rate
428         rate.sleep()
429         end = time.time()
430         #print(end - start)
431
432 # ================ LOOP ==================
433 if __name__ == '__main__':
434     try:
435         RPY_listener_and_PWM_talker()
436     except rospy.ROSInterruptException:
437         pass
```

Apéndice B

control_completo_node.py (laptop o de escritorio)

```
#!/usr/bin/env python

# ---------- DEPENDENCIAS Y BIBLIOTECAS ----------
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import time
from time import time as Time_c
import random
import math

import rospy
from std_msgs.msg import Int32MultiArray
from geometry_msgs.msg import Vector3Stamped
from geometry_msgs.msg import Pose

i = 0
datos_angulos = np.genfromtxt('/home/sigmadrian/Curso_ROS/catkin_ws/src/Bipedo_CIA/
    scripts/Tray3.txt', delimiter = '\t')

dim_trayectoria = 5029
data = np.zeros((dim_trayectoria,12)) #trayectoria
a = 120

for n in xrange(0, dim_trayectoria - a):
    data[n+a,0] = datos_angulos[n,0]
    data[n+a,1] = datos_angulos[n,1]
    data[n+a,2] = datos_angulos[n,2]
```

```
28      data[n+a,3] = datos_angulos[n,3]
29      data[n+a,4] = datos_angulos[n,4]
30      data[n+a,5] = datos_angulos[n,5]
31      data[n+a,6] = datos_angulos[n,6]
32      data[n+a,7] = datos_angulos[n,7]
33      data[n+a,8] = datos_angulos[n,8]
34      data[n+a,9] = datos_angulos[n,9]
35      data[n+a,10] = datos_angulos[n,10]
36      data[n+a,11] = datos_angulos[n,11]
37
38  datos_recabados = np.zeros((dim_trayectoria,25))
39
40  eP_pitch = np.arange(-90, 90,0.1)
41
42  eP_pitch_ENG = fuzz.trimf(eP_pitch, [-90,-90,-15])#error negativo grande
43  eP_pitch_ENP = fuzz.trimf(eP_pitch,[-60,-15,0])#error negativo pequeno
44  eP_pitch_ECC = fuzz.trapmf(eP_pitch,[-15,-5,5,15])#error cercano cero
45  eP_pitch_EPP = fuzz.trimf(eP_pitch,[0,15,60])#error positivo pequeno
46  eP_pitch_EPG = fuzz.trimf(eP_pitch, [15,90,90])#error positivo grande
47
48  eI_pitch = np.arange(-200, 200,0.1)
49
50  eI_pitch_ENG = fuzz.trimf(eI_pitch, [-200,-200,-50])#error negativo grande
51  eI_pitch_ENP = fuzz.trimf(eI_pitch,[-100,-50,0])#error negativo pequeno
52  eI_pitch_ECC = fuzz.trimf(eI_pitch,[-50,0,50])#error cercano cero
53  eI_pitch_EPP = fuzz.trimf(eI_pitch,[0,50,100])#error positivo pequeno
54  eI_pitch_EPG = fuzz.trimf(eI_pitch, [50,200,200])#error positivo grande
55
56  eD_pitch = np.arange(-300, 300,0.1)
57
58  eD_pitch_ENG = fuzz.trimf(eD_pitch, [-300,-300,-50])#error negativo grande
59  eD_pitch_ENP = fuzz.trimf(eD_pitch,[-100,-50,0])#error negativo pequeno
60  eD_pitch_ECC = fuzz.trapmf(eD_pitch,[-50,-5,5,50])#error cercano cero
61  eD_pitch_EPP = fuzz.trimf(eD_pitch,[0,50,100])#error positivo pequeno
62  eD_pitch_EPG = fuzz.trimf(eD_pitch, [50,300,300])#error positivo grande
63
64  # eP_roll = np.arange(-90, 90,0.1)
65
66  # eP_roll_ENG = fuzz.trimf(eP_roll, [-90,-90,-15])#error negativo grande
67  # eP_roll_ENP = fuzz.trimf(eP_roll,[-60,-25,0])#error negativo pequeno
68  # eP_roll_ECC = fuzz.trapmf(eP_roll,[-25,-5,5,25])#error cercano cero
69  # eP_roll_EPP = fuzz.trimf(eP_roll,[0,25,60])#error positivo pequeno
70  # eP_roll_EPG = fuzz.trimf(eP_roll, [15,90,90])#error positivo grande
71
72  KP = np.arange(0,1.5,0.1)#angulo inicial robot: 115
```

```
73
74 KP_mf1 = fuzz.trapmf(KP, [0, 0,0.2,0.4])#angulo cercano cero
75 KP_mf2 = fuzz.trimf(KP, [0.2, 0.4, 0.6])#angulo pequeno cero
76 KP_mf3 = fuzz.trimf(KP,[0.4,0.6,0.8])#angulo mitad
77 KP_mf4 = fuzz.trimf(KP, [0.6, 0.9, 1.1])#angulo pequeno mitad
78 KP_mf5 = fuzz.trapmf(KP, [0.9, 1.1, 1.5,1.5])#angulo casi final
79
80
81 KI = np.arange(0,1.5,0.1)#angulo inicial robot: 65
82
83 KI_mf1 = fuzz.trapmf(KI, [0, 0,0.2,0.4])#angulo cercano cero
84 KI_mf2 = fuzz.trimf(KI, [0.2, 0.4, 0.6])#angulo pequeno cero
85 KI_mf3 = fuzz.trimf(KI,[0.4,0.6,0.8])#angulo mitad
86 KI_mf4 = fuzz.trimf(KI, [0.6, 0.9, 1.1])#angulo pequeno mitad
87 KI_mf5 = fuzz.trapmf(KI, [0.9, 1.1, 1.5,1.5])#angulo casi final
88
89 KD = np.arange(0,0.5,0.1)#angulo inicial robot: 140
90
91 KD_mf1 = fuzz.trimf(KD,[ 0,0,0.1])#angulo cercano cero
92 KD_mf2 = fuzz.trimf(KD, [0, 0.1, 0.2])#angulo pequeno cero
93 KD_mf3 = fuzz.trimf(KD,[0.1,0.2,0.3])#angulo mitad
94 KD_mf4 = fuzz.trimf(KD, [0.2, 0.3, 0.4])#angulo pequeno mitad
95 KD_mf5 = fuzz.trapmf(KD, [0.3, 0.4, 0.5,0.5])#angulo casi final
96
97 KP_roll = np.arange(0,1.5,0.1)#angulo inicial robot: 115
98
99 KP_roll_mf1 = fuzz.trapmf(KP_roll, [0, 0,0.1,0.2])#angulo cercano cero
100 KP_roll_mf2 = fuzz.trimf(KP_roll, [0.1, 0.2, 0.4])#angulo pequeno cero
101 KP_roll_mf3 = fuzz.trimf(KP_roll,[0.2,0.4,0.8])#angulo mitad
102 KP_roll_mf4 = fuzz.trimf(KP_roll, [0.6, 0.8, 1.1])#angulo pequeno mitad
103 KP_roll_mf5 = fuzz.trapmf(KP_roll, [0.8, 1.1, 1.5,1.5])#angulo casi final
104
105 KI_roll = np.arange(0,1.5,0.1)#angulo inicial robot: 65
106
107 KI_roll_mf1 = fuzz.trapmf(KI, [0, 0,0.2,0.4])#angulo cercano cero
108 KI_roll_mf2 = fuzz.trimf(KI, [0.2, 0.4, 0.6])#angulo pequeno cero
109 KI_roll_mf3 = fuzz.trimf(KI,[0.4,0.6,0.8])#angulo mitad
110 KI_roll_mf4 = fuzz.trimf(KI, [0.6, 0.9, 1.1])#angulo pequeno mitad
111 KI_roll_mf5 = fuzz.trapmf(KI, [0.9, 1.1, 1.5,1.5])#angulo casi final
112
113 KD_roll = np.arange(0,0.5,0.1)#angulo inicial robot: 140
114
115 KD_roll_mf1 = fuzz.trimf(KD,[ 0,0,0.05])#angulo cercano cero
116 KD_roll_mf2 = fuzz.trimf(KD, [0.05, 0.1, 0.2])#angulo pequeno cero
117 KD_roll_mf3 = fuzz.trimf(KD,[0.1,0.2,0.3])#angulo mitad
```

```
KD_roll_mf4 = fuzz.trimf(KD, [0.2, 0.3, 0.4])#angulo pequeno mitad
KD_roll_mf5 = fuzz.trapmf(KD, [0.3, 0.4, 0.5,0.5])#angulo casi final

Ang_RodillaD = np.arange(0,50,1)#angulo inicial robot: 140

Ang_RodillaD_mf1 = fuzz.trapmf(Ang_RodillaD,[0,0,5,10])#angulo cercano cero
Ang_RodillaD_mf2 = fuzz.trimf(Ang_RodillaD,[5, 10,15])#angulo pequeno cero
Ang_RodillaD_mf3 = fuzz.trimf(Ang_RodillaD,[5,15,25])#angulo mitad
Ang_RodillaD_mf4 = fuzz.trimf(Ang_RodillaD,[15,30,35])#angulo pequeno mitad
Ang_RodillaD_mf5 = fuzz.trapmf(Ang_RodillaD,[30,35,50,50])#angulo casi final

Ang_RodillaI = np.arange(0,50,1)#angulo inicial robot: 140

Ang_RodillaI_mf1 = fuzz.trapmf(Ang_RodillaI,[0,0,5,10])#angulo cercano cero
Ang_RodillaI_mf2 = fuzz.trimf(Ang_RodillaI,[5, 10,15])#angulo pequeno cero
Ang_RodillaI_mf3 = fuzz.trimf(Ang_RodillaI,[5,15,25])#angulo mitad
Ang_RodillaI_mf4 = fuzz.trimf(Ang_RodillaI,[15,30,35])#angulo pequeno mitad
Ang_RodillaI_mf5 = fuzz.trapmf(Ang_RodillaI,[30,35,50,50])#angulo casi final

def ErrorP_category(error_in):
    eP_pitch_ENG_cat = fuzz.interp_membership(eP_pitch,eP_pitch_ENG,error_in)
    eP_pitch_ENP_cat = fuzz.interp_membership(eP_pitch,eP_pitch_ENP,error_in)
    eP_pitch_ECC_cat = fuzz.interp_membership(eP_pitch,eP_pitch_ECC,error_in)
    eP_pitch_EPP_cat = fuzz.interp_membership(eP_pitch,eP_pitch_EPP,error_in)
    eP_pitch_EPG_cat = fuzz.interp_membership(eP_pitch,eP_pitch_EPG,error_in)
    return dict(NG = eP_pitch_ENG_cat, NP = eP_pitch_ENP_cat, CE = eP_pitch_ECC_cat
    , PP = eP_pitch_EPP_cat, PG = eP_pitch_EPG_cat)

def ErrorI_category(error_in):
    eI_pitch_ENG_cat = fuzz.interp_membership(eI_pitch,eI_pitch_ENG,error_in)
    eI_pitch_ENP_cat = fuzz.interp_membership(eI_pitch,eI_pitch_ENP,error_in)
    eI_pitch_ECC_cat = fuzz.interp_membership(eI_pitch,eI_pitch_ECC,error_in)
    eI_pitch_EPP_cat = fuzz.interp_membership(eI_pitch,eI_pitch_EPP,error_in)
    eI_pitch_EPG_cat = fuzz.interp_membership(eI_pitch,eI_pitch_EPG,error_in)
    return dict(NG = eI_pitch_ENG_cat, NP = eI_pitch_ENP_cat, CE = eI_pitch_ECC_cat
    , PP = eI_pitch_EPP_cat, PG = eI_pitch_EPG_cat)

def ErrorD_category(error_in):
    eD_pitch_ENG_cat = fuzz.interp_membership(eD_pitch,eD_pitch_ENG,error_in)
    eD_pitch_ENP_cat = fuzz.interp_membership(eD_pitch,eD_pitch_ENP,error_in)
    eD_pitch_ECC_cat = fuzz.interp_membership(eD_pitch,eD_pitch_ECC,error_in)
    eD_pitch_EPP_cat = fuzz.interp_membership(eD_pitch,eD_pitch_EPP,error_in)
    eD_pitch_EPG_cat = fuzz.interp_membership(eD_pitch,eD_pitch_EPG,error_in)
    return dict(NG = eD_pitch_ENG_cat, NP = eD_pitch_ENP_cat, CE = eD_pitch_ECC_cat
    , PP = eD_pitch_EPP_cat, PG = eD_pitch_EPG_cat)
```

```
160
161 # def ErrorP_roll_category(error_in):
162 #   eP_roll_ENG_cat = fuzz.interp_membership(eP_roll,eP_roll_ENG,error_in)
163 #   eP_roll_ENP_cat = fuzz.interp_membership(eP_roll,eP_roll_ENP,error_in)
164 #   eP_roll_ECC_cat = fuzz.interp_membership(eP_roll,eP_roll_ECC,error_in)
165 #   eP_roll_EPP_cat = fuzz.interp_membership(eP_roll,eP_roll_EPP,error_in)
166 #   eP_roll_EPG_cat = fuzz.interp_membership(eP_roll,eP_roll_EPG,error_in)
167 #   return dict(NG = eP_roll_ENG_cat, NP = eP_roll_ENP_cat, CE = eP_roll_ECC_cat,
    PP = eP_roll_EPP_cat, PG = eP_roll_EPG_cat)
168
169
170 def marcador(tray):
171     marcador = 0
172
173     if tray > 500:#control postura
174         marcador = 1
175
176     if tray > 640:#inicio trayectoria
177         marcador = 2
178
179     if tray > 1440: #caminata 1
180         marcador = 3
181
182     if tray > 2240:#caminata 2
183         marcador = 4
184
185     if tray > 3040:#caminata 3
186         marcador = 5
187
188     if tray > 3180:#fin trayectoria
189         marcador = 6
190
191     if tray > 3680:#control postura
192         marcador = 7
193
194     return marcador
195
196 def Limites(angulo):
197     if angulo > 180:
198         angulo = 180
199     if angulo < 0:
200         angulo = 0
201     return angulo
202
203 # ------------------------- VARIABLES GLOBALES -------------------------
```

```
204 PWM = [0,0,0,0,0,0,0,0,0,0,0,0] # 12 Servomotores
205 RPY = [0.0,0.0,0.0]      # Arreglo donde se leen Roll, Pitch & Yaw
206
207 ahora_t = 0
208 ultimo_t = 0
209 dt = 0
210 error_sum_pitch = 0
211 error_sum_roll = 0
212 contador = 0
213 tiempo = 0
214 suma_tiempo = 0
215 filter_rate_pitch = 0
216 filter_rate_roll = 0
217
218 # ================== CALLBACK de ROS ================================
219 def CallBack(RPY_data):
220     # Uso de variables globales
221     global i,ahora_t, ultimo_t, dt, error_sum_pitch, error_sum_roll, contador,
    tiempo, suma_tiempo, filter_rate_pitch, filter_rate_roll
222
223     ahora_t = Time_c() # Para comenzar a medir tiempo
224     # Almacenamiento de datos de la IMU
225     RPY[0] = RPY_data.vector.x*57.2958
226     RPY[1] = RPY_data.vector.y*57.2958
227     RPY[2] = RPY_data.vector.z*57.2958
228
229     # Calculo de los errores
230     error_roll = 0 - RPY[0]
231     error_pitch = 0 - RPY[1]
232     error_rate_pitch = 0
233     error_rate_roll = 0
234     filter_rate_pitch = filter_rate_pitch + 0.1*(error_rate_pitch -
    filter_rate_pitch)
235     filter_rate_roll = filter_rate_roll +0.1*(error_rate_roll - filter_rate_roll)
236     ultimo_t = Time_c() # Para terminar de medir tiempo
237     dt = ultimo_t - ahora_t # Diferencial de tiempo
238     suma_tiempo  += dt
239     error_sum_pitch += (error_pitch*dt)
240     error_sum_roll += (error_roll*dt)
241
242     #categorias errores pitch
243     eP_rule = ErrorP_category(error_pitch)
244     eI_rule = ErrorI_category(error_sum_pitch)
245     eD_rule = ErrorD_category(filter_rate_pitch)
246
```

```
247     #categorias error roll
248     eP_roll_rule = ErrorP_category(error_roll)
249     eI_roll_rule = ErrorI_category(error_sum_roll)
250     eD_roll_rule = ErrorD_category(filter_rate_roll)
251 
252     #ERROR PITCH
253     regla1_KP = np.fmax(eP_rule['NG'],eD_rule['NG'])
254     regla2_KP = np.fmax(eP_rule['NP'],eD_rule['NP'])
255     regla3_KP = np.fmax(eP_rule['CE'],eD_rule['CE'])
256     regla4_KP = np.fmax(eP_rule['PP'],eD_rule['PP'])
257     regla5_KP = np.fmax(eP_rule['PG'],eD_rule['PG'])
258 
259     #ERROR SUMA PITCH
260     regla1_KI = np.fmax(eP_rule['NG'],eI_rule['NG'])
261     regla2_KI = np.fmax(eP_rule['NP'],eI_rule['NP'])
262     regla3_KI = np.fmax(eP_rule['CE'],eI_rule['CE'])
263     regla4_KI = np.fmax(eP_rule['PP'],eI_rule['PP'])
264     regla5_KI = np.fmax(eP_rule['PG'],eI_rule['PG'])
265 
266     #CAMBIO ERROR  PITCH
267     regla1_KD = np.fmax(eP_rule['NG'],eD_rule['NG'])
268     regla2_KD = eP_rule['NP']
269     regla3_KD = eP_rule['CE']
270     regla4_KD = eP_rule['PP']
271     regla5_KD = np.fmax(eP_rule['PG'],eD_rule['PG'])
272 
273     #ERROR ROLL
274     regla1_KP_roll = np.fmax(eP_roll_rule['NG'],eD_roll_rule['NG'])
275     regla2_KP_roll = np.fmax(eP_roll_rule['NP'],eD_roll_rule['NP'])
276     regla3_KP_roll = np.fmax(eP_roll_rule['CE'],eD_roll_rule['CE'])
277     regla4_KP_roll = np.fmax(eP_roll_rule['PP'],eD_roll_rule['PP'])
278     regla5_KP_roll = np.fmax(eP_roll_rule['PG'],eD_roll_rule['PG'])
279 
280     #ERROR SUMA ROLL
281     regla1_KI_roll = np.fmax(eP_roll_rule['NG'],eI_roll_rule['NG'])
282     regla2_KI_roll = np.fmax(eP_roll_rule['NP'],eI_roll_rule['NP'])
283     regla3_KI_roll = np.fmax(eP_roll_rule['CE'],eI_roll_rule['CE'])
284     regla4_KI_roll = np.fmax(eP_roll_rule['PP'],eI_roll_rule['PP'])
285     regla5_KI_roll = np.fmax(eP_roll_rule['PG'],eI_roll_rule['PG'])
286 
287     #CAMBIO ERROR  ROLL
288     regla1_KD_roll = np.fmax(eP_roll_rule['NG'],eD_roll_rule['NG'])
289     regla2_KD_roll = eP_roll_rule['NP']
290     regla3_KD_roll = eP_roll_rule['CE']
291     regla4_KD_roll = eP_roll_rule['PP']
```

```
292     regla5_KD_roll = np.fmax(eP_roll_rule['PG'],eD_roll_rule['PG'])
293 
294     # #ERROR ROLL cadera
295     # regla1_KP_roll_c = eP_roll_rule['NG']
296     # regla2_KP_roll_c = eP_roll_rule['NP']
297     # regla3_KP_roll_c = eP_roll_rule['CE']
298     # regla4_KP_roll_c = eP_roll_rule['PP']
299     # regla5_KP_roll_c = eP_roll_rule['PG']
300 
301     #ANGULO RODILLA
302     regla1_ang_rodillaD = eP_roll_rule['NG']
303     regla2_ang_rodillaD = eP_roll_rule['NP']
304     regla3_ang_rodillaD = eP_roll_rule['CE']
305     regla4_ang_rodillaD = eP_roll_rule['PP']
306     regla5_ang_rodillaD = eP_roll_rule['PG']
307 
308     #ANGULO RODILLA
309     regla1_ang_rodillaI = eP_roll_rule['NG']
310     regla2_ang_rodillaI = eP_roll_rule['NP']
311     regla3_ang_rodillaI = eP_roll_rule['CE']
312     regla4_ang_rodillaI = eP_roll_rule['PP']
313     regla5_ang_rodillaI = eP_roll_rule['PG']
314 
315     #Implementacion de reglas KP,KI,KD
316     regla1_KP_act = np.fmin(regla1_KP,KP_mf5)#si ENG y  entonces KP mf5 nivel alto
317     regla2_KP_act = np.fmin(regla2_KP,KP_mf3)#si ENP y entonces KP mf2, nivel bajo
318     regla3_KP_act = np.fmin(regla3_KP,KP_mf1)#si ECC y entonces KP mf3
319     regla4_KP_act = np.fmin(regla4_KP,KP_mf3)#si EPP y entonces KP mf4
320     regla5_KP_act = np.fmin(regla5_KP,KP_mf5)#si EPG y entonces KP mf5
321 
322     regla1_KI_act = np.fmin(regla1_KI,KI_mf1)#si ENG y ENG entonces KI mf1
323     regla2_KI_act = np.fmin(regla2_KI,KI_mf3)#si ENP y entonces KI mf2
324     regla3_KI_act = np.fmin(regla3_KI,KI_mf4)#si ECC y entonces KI mf3
325     regla4_KI_act = np.fmin(regla4_KI,KI_mf3)#si EPP y entonces KI mf4
326     regla5_KI_act = np.fmin(regla5_KI,KI_mf1)#si EPG y entonces KI mf5
327 
328     regla1_KD_act = np.fmin(regla1_KD,KD_mf4)#si ENG y ENG entonces KD mf1
329     regla2_KD_act = np.fmin(regla2_KD,KD_mf2)#si ENP y entonces KD mf2
330     regla3_KD_act = np.fmin(regla3_KD,KD_mf1)#si ECC y entonces KD mf3
331     regla4_KD_act = np.fmin(regla4_KD,KD_mf2)#si EPP y entonces KD mf4
332     regla5_KD_act = np.fmin(regla5_KD,KD_mf4)#si EPG y entonces KD mf5
333 
334     regla1_KP_roll_act = np.fmin(regla1_KP_roll,KP_roll_mf2)#si ENG y  entonces KP
    mf5 nivel alto
335     regla2_KP_roll_act = np.fmin(regla2_KP_roll,KP_roll_mf1)#si ENP y entonces KP
```

```
    mf2, nivel bajo
336     regla3_KP_roll_act = np.fmin(regla3_KP_roll,KP_roll_mf1)#si ECC y entonces KP
    mf3
337     regla4_KP_roll_act = np.fmin(regla4_KP_roll,KP_roll_mf1)#si EPP y entonces KP
    mf4
338     regla5_KP_roll_act = np.fmin(regla5_KP_roll,KP_roll_mf2)#si EPG y entonces KP
    mf5
339
340     regla1_KI_roll_act = np.fmin(regla1_KI_roll,KI_roll_mf1)#si ENG y  entonces KP
    mf5 nivel alto
341     regla2_KI_roll_act = np.fmin(regla2_KI_roll,KI_roll_mf2)#si ENP y entonces KP
    mf2, nivel bajo
342     regla3_KI_roll_act = np.fmin(regla3_KI_roll,KI_roll_mf3)#si ECC y entonces KP
    mf3
343     regla4_KI_roll_act = np.fmin(regla4_KI_roll,KI_roll_mf2)#si EPP y entonces KP
    mf4
344     regla5_KI_roll_act = np.fmin(regla5_KI_roll,KI_roll_mf1)#si EPG y entonces KP
    mf5
345
346     regla1_KD_roll_act = np.fmin(regla1_KD_roll,KD_roll_mf2)#si ENG y  entonces KP
    mf5 nivel alto
347     regla2_KD_roll_act = np.fmin(regla2_KD_roll,KD_roll_mf1)#si ENP y entonces KP
    mf2, nivel bajo
348     regla3_KD_roll_act = np.fmin(regla3_KD_roll,KD_roll_mf1)#si ECC y entonces KP
    mf3
349     regla4_KD_roll_act = np.fmin(regla4_KD_roll,KD_roll_mf1)#si EPP y entonces KP
    mf4
350     regla5_KD_roll_act = np.fmin(regla5_KD_roll,KD_roll_mf2)#si EPG y entonces KP
    mf5
351
352     regla1_ang_rodillaD_act = np.fmin(regla1_ang_rodillaD,Ang_RodillaD_mf5)#si ENG
    y ENG entonces KD mf1
353     regla2_ang_rodillaD_act = np.fmin(regla2_ang_rodillaD,Ang_RodillaD_mf3)#si ENP
    y entonces KD mf2
354     regla3_ang_rodillaD_act = np.fmin(regla3_ang_rodillaD,Ang_RodillaD_mf1)#si ECC
    y entonces KD mf3
355     regla4_ang_rodillaD_act = np.fmin(regla4_ang_rodillaD,Ang_RodillaD_mf3)#si EPP
    y entonces KD mf4
356     regla5_ang_rodillaD_act = np.fmin(regla5_ang_rodillaD,Ang_RodillaD_mf5)#si EPG
    y entonces KD mf5
357
358     regla1_ang_rodillaI_act = np.fmin(regla1_ang_rodillaI,Ang_RodillaI_mf5)#si ENG
    y ENG entonces KD mf1
359     regla2_ang_rodillaI_act = np.fmin(regla2_ang_rodillaI,Ang_RodillaI_mf3)#si ENP
    y entonces KD mf2
```

```
360     regla3_ang_rodillaI_act = np.fmin(regla3_ang_rodillaI, Ang_RodillaI_mf1)#si ECC
    y entonces KD mf3
361     regla4_ang_rodillaI_act = np.fmin(regla4_ang_rodillaI, Ang_RodillaI_mf3)#si EPP
    y entonces KD mf4
362     regla5_ang_rodillaI_act = np.fmin(regla5_ang_rodillaI, Ang_RodillaI_mf5)#si EPG
    y entonces KD mf5
363
364     SumaMembresiasKP = np.fmax(regla1_KP_act, np.fmax(regla2_KP_act, np.fmax(
    regla3_KP_act, np.fmax(regla4_KP_act, regla5_KP_act))))
365     SumaMembresiasKI =  np.fmax(regla1_KI_act, np.fmax(regla2_KI_act, np.fmax(
    regla3_KI_act, np.fmax(regla4_KI_act, regla5_KI_act))))
366     SumaMembresiasKD = np.fmax(regla1_KD_act, np.fmax(regla2_KD_act, np.fmax(
    regla3_KD_act, np.fmax(regla4_KD_act, regla5_KD_act))))
367
368     SumaMembresiasKP_roll = np.fmax(regla1_KP_roll_act, np.fmax(regla2_KP_roll_act,
    np.fmax(regla3_KP_roll_act, np.fmax(regla4_KP_roll_act, regla5_KP_roll_act))))
369     SumaMembresiasKI_roll = np.fmax(regla1_KI_roll_act, np.fmax(regla2_KI_roll_act,
    np.fmax(regla3_KI_roll_act, np.fmax(regla4_KI_roll_act, regla5_KI_roll_act))))
370     SumaMembresiasKD_roll = np.fmax(regla1_KD_roll_act, np.fmax(regla2_KD_roll_act,
    np.fmax(regla3_KD_roll_act, np.fmax(regla4_KD_roll_act, regla5_KD_roll_act))))
371
372     SumaMembresias_a_D = np.fmax(regla1_ang_rodillaD_act, np.fmax(
    regla2_ang_rodillaD_act, np.fmax(regla3_ang_rodillaD_act, np.fmax(
    regla4_ang_rodillaD_act, regla5_ang_rodillaD_act))))
373     SumaMembresias_a_I = np.fmax(regla1_ang_rodillaI_act, np.fmax(
    regla2_ang_rodillaI_act, np.fmax(regla3_ang_rodillaI_act, np.fmax(
    regla4_ang_rodillaI_act, regla5_ang_rodillaI_act))))
374
375     #defuzzification, resultado:
376     Kp_pitch = fuzz.centroid(KP, SumaMembresiasKP)
377     Ki_pitch = fuzz.centroid(KI, SumaMembresiasKI)
378     Kd_pitch = fuzz.centroid(KD, SumaMembresiasKD)
379
380     Kp_roll = fuzz.centroid(KP_roll, SumaMembresiasKP_roll)
381     Ki_roll = fuzz.centroid(KI_roll, SumaMembresiasKI_roll)
382     Kd_roll = fuzz.centroid(KD_roll, SumaMembresiasKD_roll)
383
384
385     a_D = fuzz.centroid(Ang_RodillaD, SumaMembresias_a_D)
386     a_I = fuzz.centroid(Ang_RodillaI, SumaMembresias_a_I)
387
388     #ADQUISiCION DE DATOS, GRAFICAS
389     '''datos_recabados[i,0] = suma_tiempo
390     datos_recabados[i,1] = error_pitch
391     datos_recabados[i,2] = error_roll
```

```
392     datos_recabados[i,3] = error_rate_pitch
393     datos_recabados[i,4] = error_rate_roll
394     datos_recabados[i,5] = filter_rate_pitch
395     datos_recabados[i,6] = filter_rate_roll
396     datos_recabados[i,7] = error_sum_pitch
397     datos_recabados[i,8] = error_sum_roll
398     datos_recabados[i,9] = Kp_pitch
399     datos_recabados[i,10] = Ki_pitch
400     datos_recabados[i,11] = Kd_pitch
401     datos_recabados[i,12] = Kp_roll
402     datos_recabados[i,13] = Ki_roll
403     datos_recabados[i,14] = Kd_roll'''
404 
405     if (i < a + 1):
406         servo11 = 90
407         servo12 = 90
408         servo21 = Limites(int(90 + Kp_roll*error_roll + Ki_roll*error_sum_roll +
    Kd_roll*error_rate_roll))
409         servo22 = Limites(int(90 + Kp_roll*error_roll + Ki_roll*error_sum_roll +
    Kd_roll*error_rate_roll))
410         servo31 = Limites(int(150 - Kp_pitch*error_pitch - Ki_pitch*error_sum_pitch
     - Kd_pitch*error_rate_pitch - a_D))#cadera PID
411         servo32 = Limites(int(45 + Kp_pitch*error_pitch + Ki_pitch*error_sum_pitch
    + Kd_pitch*error_rate_pitch +  a_I))#cadera PID
412         servo41 = Limites(int(65 - Kp_pitch*error_pitch - Ki_pitch*error_sum_pitch
    - Kd_pitch*error_rate_pitch - a_D))#Rodilla
413         servo42 = Limites(int(120 + Kp_pitch*error_pitch + Ki_pitch*error_sum_pitch
     + Kd_pitch*error_rate_pitch + a_I))#Rodilla
414         servo51 = Limites(int(120 - Kp_pitch*error_pitch - Ki_pitch*error_sum_pitch
     - Kd_pitch*error_rate_pitch - a_D))#Tobillo PID
415         servo52 = Limites(int(65 + Kp_pitch*error_pitch + Ki_pitch*error_sum_pitch
    + Kd_pitch*error_rate_pitch + a_I))#Tobillo PID
416         servo61 = Limites(int(90 + Kp_roll*error_roll+ Ki_roll*error_sum_roll +
    Kd_roll*error_rate_roll))
417         servo62 = Limites(int(95 + Kp_roll*error_roll+ Ki_roll*error_sum_roll +
    Kd_roll*error_rate_roll))
418     if(i > a ):
419         servo12 = int(data[i,0])
420         servo11 = int(data[i,1])
421         servo22 = Limites(int(data[i,2] + Kp_roll*error_roll + Ki_roll*
    error_sum_roll + Kd_roll*error_rate_roll))
422         servo21 = Limites(int(data[i,3] + Kp_roll*error_roll + Ki_roll*
    error_sum_roll + Kd_roll*error_rate_roll))
423         servo32 = Limites(int(data[i,4] + Kp_pitch*error_pitch + Ki_pitch*
    error_sum_pitch + Kd_pitch*error_rate_pitch ))#+  a_I))#cadera PID
```

```
424         servo31 = Limites(int(data[i,5] - Kp_pitch*error_pitch - Ki_pitch*
    error_sum_pitch - Kd_pitch*error_rate_pitch ))#- a_D))#cadera PID
425         servo42 = Limites(int(data[i,6] + Kp_pitch*error_pitch + Ki_pitch*
    error_sum_pitch + Kd_pitch*error_rate_pitch ))#+ a_I))#Rodilla
426         servo41 = Limites(int(data[i,7] - Kp_pitch*error_pitch - Ki_pitch*
    error_sum_pitch - Kd_pitch*error_rate_pitch ))#- a_D))#Rodilla
427         servo52 = Limites(int(data[i,8] + Kp_pitch*error_pitch + Ki_pitch*
    error_sum_pitch + Kd_pitch*error_rate_pitch ))#+ a_I))#Tobillo PID
428         servo51 = Limites(int(data[i,9] - Kp_pitch*error_pitch - Ki_pitch*
    error_sum_pitch - Kd_pitch*error_rate_pitch ))#- a_D))#Tobillo PID
429         servo62 = Limites(int(data[i,10] + Kp_roll*error_roll+ Ki_roll*
    error_sum_roll + Kd_roll*error_rate_roll))
430         servo61 = Limites(int(data[i,11] + Kp_roll*error_roll+ Ki_roll*
    error_sum_roll + Kd_roll*error_rate_roll))
431
432     '''datos_recabados[i,15] = servo22
433     datos_recabados[i,16] = servo21
434     datos_recabados[i,17] = servo32
435     datos_recabados[i,18] = servo31
436     datos_recabados[i,19] = servo42
437     datos_recabados[i,20] = servo41
438     datos_recabados[i,21] = servo52
439     datos_recabados[i,22] = servo51
440     datos_recabados[i,23] = servo62
441     datos_recabados[i,24] = servo61'''
442
443     PWM[0] = servo11+2              # servo11
444     PWM[1] = servo12              # servo12
445     PWM[2] = servo21              # servo21
446     PWM[3] = servo22+7              # servo22
447     PWM[4] = servo31+5#cadera PID  # servo31
448     PWM[5] = servo32-2#cadera PID  # servo32
449     PWM[6] = servo41-7#Rodilla     # servo41
450     PWM[7] = servo42+7#Rodilla     # servo42
451     PWM[8] = servo51#Tobillo PID # servo51
452     PWM[9] = servo52#Tobillo PID # servo52
453     PWM[10] = servo61-2             # servo61
454     PWM[11] = servo62+10             # servo62
455
456     ''PWM[0] = servo11              # servo11
457     PWM[1] = servo12              # servo12
458     PWM[2] = servo21              # servo21
459     PWM[3] = servo22              # servo22
460     PWM[4] = servo31#cadera PID  # servo31
461     PWM[5] = servo32#cadera PID  # servo32
```

```
462     PWM[6] = servo41#Rodilla      # servo41
463     PWM[7] = servo42#Rodilla      # servo42
464     PWM[8] = servo51#Tobillo PID # servo51
465     PWM[9] = servo52#Tobillo PID # servo52
466     PWM[10] = servo61             # servo61
467     PWM[11] = servo62             # servo62'''
468 
469     i = i + 1 # Para ser utilizado por las matrices de arriba
470 
471 #==================== Publicador y Subscriptor ====================
472 def Bipedo_Publisher_and_Subscriber():
473 
474     # Nombre del nodo, false para que no despliegue numero aleatorio
475     rospy.init_node('control_completo_node', anonymous=False)
476     print("Inicializando control_completo_node...")
477 
478     # Definicion del publicador
479     pub = rospy.Publisher('/servos_topic', Int32MultiArray, queue_size=10)
480     pub_plot = rospy.Publisher('/servos_plot_compu', Pose, queue_size=10)
481 
482     # Definicion del subscriptor
483     rospy.Subscriber('/imu/rpy', Vector3Stamped, CallBack)
484     #rospy.spin() <-------- este es remplazado por while not rospy.is_shutdown()
485 
486     # Velocidad del programa
487     rate = rospy.Rate(500) # (10) = 10[Hz]
488 
489     # Arreglo de 12 servos donde se almacenaran los datos a publicar
490     angulos = Int32MultiArray()
491     angulos_plot = Pose()
492 
493     # Angulos iniciales
494     angulos.data = [90,90,90,87,138,60,60,120,120,60,87,87];
495     angulos_plot.position.x = 0
496 
497     #============== WHILE LOOP ============================
498     while not rospy.is_shutdown():  # Confirma que todo esta bien
499 
500         # Uso de variables globales
501         global i,PWM, error_roll, error_pitch, error_sum, filter_rate, Kp, Ki,
    datos_recabados
502 
503         # Almacenamiento en arreglo del publicador
504         if RPY[0] == 0 and RPY[1] == 0:
505             angulos.data = [90,90,90,87,141,57,60,120,120,60,87,87];
```

```
506         else:
507             angulos.data[0] = PWM[0]
508             angulos.data[1] = PWM[1]
509             angulos.data[2] = PWM[2]
510             angulos.data[3] = PWM[3]
511             angulos.data[4] = PWM[4]
512             angulos.data[5] = PWM[5]
513             angulos.data[6] = PWM[6]
514             angulos.data[7] = PWM[7]
515             angulos.data[8] = PWM[8]
516             angulos.data[9] = PWM[9]
517             angulos.data[10] = PWM[10]
518             angulos.data[11] = PWM[11]
519 
520             angulos_plot.position.x = angulos.data[10] # el servo de posicion 10 de
    muestra
521 
522         # Impresion de datos de calculo
523         #print "eR: ",error_roll,"eP: ", error_pitch," eI: ",error_sum_pitch," eD:
    ",filter_rate_pitch, " Kp_pitch: ",Kp_pitch, " Ki_pitch: ",Ki_pitch,"contador",i
    #, " Marcador: ",marcador(i)
524         #print "Roll: ", RPY[0], " Pitch: ", RPY[1]
525         #print angulos.data[0]," ",angulos.data[1]," ",angulos.data[2]," ",angulos.
    data[3]," ",
526 
527         # Publicacion del mensaje
528         pub.publish(angulos)
529         #pub_plot.publish(angulos_plot) # Publicador de graficacion
530         # Delay del programa definido en el rate
531         rate.sleep()
532 
533         #Sustituye el for xrange del antiguo control_completo, para que pueda
    salirse
534         # del programa una vez completado la dimension de la trayectoria
535         if i >= (dim_trayectoria - 1):
536           print "TRAYECTORIA TERMINADA"
537         exit()
538 
539 # ================ LOOP ====================
540 if __name__ == '__main__':
541     try:
542         Bipedo_Publisher_and_Subscriber()
543     except rospy.ROSInterruptException:
544         pass
```

Apéndice C

tiva_servos_node.ino (Tiva-C)

```
// ---------- BIBLIOTECAS ----------
#include <ros.h>
#include <std_msgs/Int32MultiArray.h>
//#include <geometry_msgs/Pose.h> // Para rqt_plot
#include <Servoh.h>
// ---------------------------------

// ---------- PINES A UTILIZAR ----------
// CADERA
#define s11_pin PD_7    // izquierda  YAW
#define s12_pin PF_4    // derecha YAW
#define s21_pin PC_7    // izquierda ROLL
#define s22_pin PD_6    // derecha ROLL
#define s31_pin PC_5    // izquierda PITCH
#define s32_pin PC_6    // derecha PITCH

// RODILLA
#define s41_pin PD_3    // izquierda PITCH
#define s42_pin PD_2    // derecha PITCH

// TOBILLO
#define s51_pin PE_2    // izquierda PITCH
#define s52_pin PE_1    // derecha PITCH
#define s61_pin PF_1    // izquierda ROLL
#define s62_pin PE_3    // derecha ROLL
// ---------------------------------

// ---------- Variables y Objetos Globales ----------

Servo servos[12];   // Arreglo de objetos de los 12 servos

```

```
32 // Arreglo de angulos de los servos, calculados con los datos la IMU en Rasp
33 long angulos[12];
34 //long angulo_plot = 0;
35 int angulos_iniciales[12] = {90,90,90,87,141,57,60,120,120,60,87,87};
36 //int angulos_iniciales[12] = {90,90,90,87,138,60,60,120,120,60,87,87};   //
    anteriores
37 // ---------------------------------------------------------
38
39 // ************** CALLBACK ************************
40 void messageCb(const std_msgs::Int32MultiArray& potencia_msg){
41   for(int i = 0; i < 12; i++)
42     angulos[i] = potencia_msg.data[i];
43
44   // Nota**: Nunca deben haber valores negativos en la escritura de angulos
45 }
46 // ************************************************
47
48 // Crea un arreglo para data console
49 //geometry_msgs::Pose data_plot;    // DESCOMENTAR ESTA PARA RQT_PLOT
50
51 // Publicador
52 //ros::Publisher publisher("/servo_plot_tiva", &data_plot);     // DESCOMENTAR ESTA
    PARA RQT_PLOT
53
54 // Suscriptor
55 ros::Subscriber<std_msgs::Int32MultiArray> sub("/servos_topic", &messageCb );
56
57 ros::NodeHandle nh;
58
59 // ================ VOID SETUP ======================
60 void setup() {
61   nh.getHardware()->setBaud(500000);  //500,000 baudrate
62
63   nh.initNode();       // Inicializa el nodo
64
65   //Advertir a nh que voy a publicar
66   //nh.advertise(publisher);     // DESCOMENTAR ESTA PARA RQT_PLOT
67
68   nh.subscribe(sub);  // Se suscribe
69
70   // Activacion de Servomotores
71   servos[0].attach(s11_pin,900,2100);
72   servos[1].attach(s12_pin,900,2100);
73   servos[2].attach(s21_pin,900,2100);
74   servos[3].attach(s22_pin,900,2100);
```

```
75    servos[4].attach(s31_pin,900,2100);
76    servos[5].attach(s32_pin,900,2100);
77    servos[6].attach(s41_pin,900,2100);
78    servos[7].attach(s42_pin,900,2100);
79    servos[8].attach(s51_pin,900,2100);
80    servos[9].attach(s52_pin,900,2100);
81    servos[10].attach(s61_pin,900,2100);
82    servos[11].attach(s62_pin,750,2250);
83    //<---------750-2250 micro-sec servo hs 5485 mg
84    // (instruccion ya establecida por Allen)
85
86    // Posicion inicial Servomotores
87    for(int i = 0; i < 12; i++)
88    {
89      servos[i].write(angulos_iniciales[i]);
90
91      // De igualan estos por si no existe ningun mensaje hasta entonces
92      // y el robot se mantenga en su posicion inicial
93      angulos[i] = angulos_iniciales[i];
94    }
95  }
96  // ================================================
97
98  // ============== VOID LOOP ======================
99  void loop()
100 {
101   for(int i = 0; i < 12; i++)
102     servos[i].write(angulos[i]);
103
104   //data_plot.position.x = angulos[10];     // DESCOMENTAR ESTA PARA RQT_PLOT    PIE
        IZQUIERDO
105
106   //publisher.publish(&data_plot);    // DESCOMENTAR ESTA PARA RQT_PLOT
107
108   nh.spinOnce();
109 }
110 // ================================================
```

Apéndice D

sensores_node.py (Raspberry)

```
#!/usr/bin/env python

import rospy    # Esto es para podernos comunicar con ROS mediante python
import serial   # Necesaria para leer por serial
import time
from std_msgs.msg import Float32  # Voy a publicar en un topico el tipo de dato Float32
from std_msgs.msg import Float32MultiArray  # Voy a publicar en un topico el tipo de dato Float32

flex_array = [0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
pots_array = [0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
RPY = [0.0,0.0,0.0]     # Arreglo donde se leen Roll, Pitch & Yaw

# CAMBIAR PUERTO dependiendo del disponible
Arduino = serial.Serial("/dev/Arduino-UNO",115200)       # Crea el puerto serial a cierta velocidad baudrate

print "Initializing Sensores_node"

def talker():
  # Para ignorar la basura inicial del serial:
   for i in range(1,12):
     data = Arduino.readline()

     while not rospy.is_shutdown():      # Este es el que nos va a devolver si todo esta bien, es como el while ros ok en C++
     data = Arduino.readline()       # Guardamos en una variable, regla de 3 para que de una vuelta completa

     DataSensors = data.split("#")
```

```
27      DataFlex = DataSensors[0]
28      DataPots = DataSensors[1]
29      DataYaw = DataSensors[2]
30
31      FlexSensors = DataFlex.split(",")
32      PotsSensors = DataPots.split(",")
33
34      flex_array[0] = float(FlexSensors[0])
35      flex_array[1] = float(FlexSensors[1])
36      flex_array[2] = float(FlexSensors[2])
37      flex_array[3] = float(FlexSensors[3])
38      flex_array[4] = float(FlexSensors[4])
39      flex_array[5] = float(FlexSensors[5])
40      flex_array[6] = float(FlexSensors[6])
41      flex_array[7] = float(FlexSensors[7])
42      flex_array[8] = float(FlexSensors[8])
43      flex_array[9] = float(FlexSensors[9])
44      flex_array[10] = float(FlexSensors[10])
45            flex_array[11] = float(FlexSensors[11])
46      flex_array[12] = float(FlexSensors[12])
47      flex_array[13] = float(FlexSensors[13])
48      flex_array[14] = float(FlexSensors[14])
49      flex_array[15] = float(FlexSensors[15])
50
51      pots_array[0] = float(PotsSensors[0])
52      pots_array[1] = float(PotsSensors[1])
53      pots_array[2] = float(PotsSensors[2])
54      pots_array[3] = float(PotsSensors[3])
55      pots_array[4] = float(PotsSensors[4])
56      pots_array[5] = float(PotsSensors[5])
57      pots_array[6] = float(PotsSensors[6])
58      pots_array[7] = float(PotsSensors[7])
59      pots_array[8] = float(PotsSensors[8])
60      pots_array[9] = float(PotsSensors[9])
61      pots_array[10] = float(PotsSensors[10])
62      pots_array[11] = float(PotsSensors[11])
63      pots_array[12] = float(PotsSensors[12])
64      pots_array[13] = float(PotsSensors[13])
65      pots_array[14] = float(PotsSensors[14])
66      pots_array[15] = float(PotsSensors[15])
67
68      flex_data.data = flex_array
69      pots_data.data = pots_array
70
71     #print " DataFlex: " + DataFlex + " DataPots: " + DataPots + " Yaw: " + DataYaw
```

```
72
73      #rospy.loginfo(data)              # Para mostrar lo que le esta llegando
74      pubFlex.publish(flex_data)         # Conversion a flotante
75      pubPots.publish(pots_data)
76      pubYaw.publish(float(DataYaw))
77
78    rospy.sleep(0.0001)                # Lo que duerme
79
80  if __name__ == '__main__':
81    try:
82      # Publicador
83      pubFlex = rospy.Publisher("/flex",Float32MultiArray,queue_size=1);
84      pubPots = rospy.Publisher("/pots",Float32MultiArray,queue_size=1);
85      pubYaw = rospy.Publisher("/yaw",Float32,queue_size=1);
86
87      flex_data = Float32MultiArray();
88      pots_data = Float32MultiArray();
89
90      rospy.init_node("Sensores_node");   # Inicializa el nodo, es con el que ROS va
        a identificar el nodo, puede ser distinto el nombre al del nodo, o igual. El
        punto y coma no es necesario.
91      talker()
92    except rospy.ROSInterruptException:
93      pass
```

Apéndice E

flex_pots_yaw_BIPEDO.ino (Arduino-UNO)

```
float angulo_z = 0.0;

// control pins output table in array form
// see truth table on page 2 of TI 74HC4067 data sheet
// connect 74HC4067 S0~S3 to Arduino D7~D4 respectively
// connect 74HC4067 pin 1 to Arduino A0
byte controlPins_Flex[] =
                    {B00000000,
                    B10000000,
                    B01000000,
                    B11000000,
                    B00100000,
                    B10100000,
                    B01100000,
                    B11100000,
                    B00010000,
                    B10010000,
                    B01010000,
                    B11010000,
                    B00110000,
                    B10110000,
                    B01110000,
                    B11110000};

byte controlPins_Pots[] =
                    {B000000,
                    B100000,
                    B010000,
```

```
                    B110000,
                    B001000,
                    B101000,
                    B011000,
                    B111000,
                    B000100,
                    B100100,
                    B010100,
                    B110100,
                    B001100,
                    B101100,
                    B011100,
                    B111100};

// holds incoming values from 74HC4067
byte muxValues_Flex[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
byte muxValues_Pots[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

// ================================================================
// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#include "MPU6050.h"

// Arduino Wire library   is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// class default I2C address is 0x68
MPU6050 mpu(0x68);

// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO.
#define OUTPUT_READABLE_YAWPITCHROLL

#define LED_PIN 13
bool blinkState = false;

// ================================ VARIABLES GLOBALES
   ================================================
bool dmpReady = false;  // set true if DMP init was successful
```

```
73 uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
74 uint8_t devStatus;      // return status after each device operation (0 = success,
       !0 = error)
75 uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
76 uint16_t fifoCount;     // count of all bytes currently in FIFO
77 uint8_t fifoBuffer[64]; // FIFO storage buffer
78
79 // orientation/motion vars
80 Quaternion q;           // [w, x, y, z]         quaternion container
81 VectorInt16 aa;         // [x, y, z]            accel sensor measurements
82 VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor
       measurements
83 VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor
       measurements
84 VectorFloat gravity;    // [x, y, z]            gravity vector
85 float euler[3];         // [psi, theta, phi]    Euler angle container
86 float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and
       gravity vector
87
88 // packet structure for InvenSense teapot demo
89 uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n'
       };
90
91 // ================================================ INTERRUPCION
       ================================================
92
93 volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has
       gone high
94 void dmpDataReady() {
95     mpuInterrupt = true;
96 }
97
98 // ================================================ VOID SETUP
       ================================================
99 void setup() {
100     // join I2C bus (I2Cdev library doesn't do this automatically)
101     #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
102         Wire.begin();
103         TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
104     #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
105         Fastwire::setup(400, true);
106     #endif
107
108     Serial.begin(115200);
109
```

```
110     // Para las direcciones de los multiplexores
111     DDRD = B11111011;
112     DDRB = B111111;
113 
114     while (!Serial); // wait for Leonardo enumeration, others continue immediately
115 
116     mpu.initialize();    // Inicializacion del MPU
117 
118     // load and configure the DMP
119     //Serial.println(F("Initializing DMP..."));
120     devStatus = mpu.dmpInitialize();
121 
122     // Offsets
123     mpu.setXGyroOffset(220);
124     mpu.setYGyroOffset(76);
125     mpu.setZGyroOffset(-85);
126     mpu.setZAccelOffset(1788); // 1688 default de fabrica
127 
128     // make sure it worked (returns 0 if so)
129     if (devStatus == 0) {
130         // turn on the DMP, now that it's ready
131         //Serial.println(F("Enabling DMP..."));
132         mpu.setDMPEnabled(true);
133 
134         // enable Arduino interrupt detection
135         //Serial.println(F("Enabling interrupt detection (Arduino external
    interrupt 0)..."));
136         attachInterrupt(0, dmpDataReady, RISING);
137         mpuIntStatus = mpu.getIntStatus();
138 
139         // set our DMP Ready flag so the main loop() function knows it's okay to
    use it
140         //Serial.println(F("DMP ready! Waiting for first interrupt..."));
141         dmpReady = true;
142 
143         // get expected DMP packet size for later comparison
144         packetSize = mpu.dmpGetFIFOPacketSize();
145     } else {
146         // ERROR!
147         // 1 = initial memory load failed
148         // 2 = DMP configuration updates failed
149         // (if it's going to break, usually the code will be 1)
150         //Serial.print(F("DMP Initialization failed (code "));
151         //Serial.print(devStatus);
152         //Serial.println(F(")"));
```

```
153     }
154     pinMode(LED_PIN, OUTPUT);
155 }
156 
157 void setPin(int outputPin)
158 // function to select pin on 74HC4067
159 {
160   PORTD  = controlPins_Flex[outputPin];
161   PORTB  = controlPins_Pots[outputPin];
162 }
163 
164 // ================================== VOID LOOP
    ==========================================
165 
166 void loop() {
167   // if programming failed, don't try to do anything
168   if (!dmpReady) return;  //*DESCOMENTAR*
169 
170   // wait for MPU interrupt or extra packet(s) available
171   while (!mpuInterrupt && fifoCount < packetSize){}
172 
173   // reset interrupt flag and get INT_STATUS byte
174   mpuInterrupt = false;
175   mpuIntStatus = mpu.getIntStatus();
176 
177   // get current FIFO count
178   fifoCount = mpu.getFIFOCount();
179 
180   // check for overflow (this should never happen unless our code is too
     inefficient)
181   if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
182       // reset so we can continue cleanly
183       mpu.resetFIFO();
184       //Serial.println(F("FIFO overflow!"));
185 
186   // otherwise, check for DMP data ready interrupt (this should happen frequently)
187   } else if(mpuIntStatus & 0x02) {
188       // wait for correct available data length, should be a VERY short wait
189       while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
190       // read a packet from FIFO
191       mpu.getFIFOBytes(fifoBuffer, packetSize);
192   }
193       // track FIFO count here in case there is > 1 packet available
194      // (this lets us immediately read more without waiting for an interrupt)
195       fifoCount -= packetSize;
```

```
196
197     #ifdef OUTPUT_READABLE_YAWPITCHROLL
198         // display Euler angles in degrees
199         mpu.dmpGetQuaternion(&q, fifoBuffer);
200         mpu.dmpGetGravity(&gravity, &q);
201         mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
202         angulo_z= ypr[0] * 180/M_PI;        //  ypr[0] es en z
203
204         if( angulo_z < 0 )
205           angulo_z= 360+angulo_z;
206
207         angulo_z = 360 - angulo_z;
208         for (int i = 0; i < 16; i++)
209         {
210           setPin(i); // choose an input pin on the 74HC4067
211           muxValues_Flex[i]=analogRead(0); // read the vlaue on that pin and store
    in array
212           muxValues_Pots[i]=analogRead(1); // read the vlaue on that pin and store
    in array
213         }
214
215         // Despliegue de valores
216         for (int i = 0; i < 15; i++)
217         {
218           Serial.print(muxValues_Flex[i]);
219           Serial.print(",");
220         }
221         Serial.print(muxValues_Flex[15]);
222         Serial.print("#");
223
224         for (int i = 0; i < 15; i++)
225         {
226           Serial.print(muxValues_Pots[i]);
227           Serial.print(",");
228         }
229         Serial.print(muxValues_Pots[15]);
230         Serial.print("#");
231
232         Serial.print(angulo_z,3);
233         Serial.println("#");
234
235     #endif
236
237     // blink LED to indicate activity
238     blinkState = !blinkState;
```

```
239     digitalWrite(LED_PIN, blinkState);
240 }
```

Apéndice F

.bashrc (laptop o de escritorio)

Agregar las siguientes líneas de código al archivo **.bashrc** que se puede editar con el comando:

subl ~/.bashrc (o el editor de texto a la mano en la computadora)

```
source /opt/ros/kinetic/setup.bash
source ~/Curso_ROS/catkin_ws/devel/setup.bash
alias cm="catkin_make -C ~/Curso_ROS/catkin_ws"
#Lo siguiente es solo para que la compu se conecte directamente con el BIPEDO
export ROS_MASTER_URI=http://(PONER AQUI IP DE LA RASPBERRY):11311
export ROS_IP=(PONER AQUI IP DE LA COMPUTADORA)
```

Apéndice G

.bashrc (Raspberry)

Agregar las siguientes líneas de código al archivo **.bashrc** que se puede editar con el comando:

pluma ~/.bashrc (o el editor de texto a la raspberry)

```
source /opt/ros/kinetic/setup.bash
source ~/Curso_ROS/catkin_ws/devel/setup.bash
alias cm="catkin_make -C ~/Curso_ROS/catkin_ws"
#Lo siguiente es solo para que el BIPEDO se conecte con la compu
export ROS_MASTER_URI=http://(PONER AQUI IP DE LA RASPBERRY):11311
export ROS_IP=(PONER AQUI IP DE LA RASPBERRY)
```

Apéndice H

bipedo.launch (Raspberry)

```
<launch>
  <node pkg="um7" name="um7_driver" type="um7_driver" output="screen">
    <param name="port" value="/dev/ttyS0"/>
  </node>

  <node pkg="Bipedo" name="sensores_node" type="sensores_node.py" output="screen">
    <param name="port" value="/dev/Arduino-UNO"/>
    <param name="baud" value="115200"/>
  </node>

  <group ns="rosserial">
        <node pkg="rosserial_python" name="serial_node" type="serial_node.py"
    output="screen">
                <param name="port" value="/dev/Tiva-C"/>
                <param name="baud" value="500000"/>
        </node>
    </group>
</launch>
```

Apéndice I

Bipedo.sh (Raspberry)

```
#!/bin/bash

cd
source /opt/ros/kinetic/setup.bash
source ~/catkin_ws/devel/setup.bash

sudo chmod 777 /dev/Arduino-UNO
sudo chmod 777 /dev/Tiva-C
sudo chmod 777 /dev/ttyS0

exit 0
```

Apéndice J

rc.local (Raspberry)

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

sleep 12 &
cd ~bipedoscout01/Scripts/
./Bipedo.sh &

exit 0
```

Apéndice K

scratch (Raspberry)

El archivo *scratch* se abre con el comando:

sudo visudo

```
# Allow members of group gpio to execute scratch and sqweak
%gpio ALL=NOPASSWD: /usr/bin/pigpiod_wrapper
%gpio ALL=NOPASSWD: /usr/bin/scratch
%gpio ALL=NOPASSWD: /usr/bin/squeak
bipedoscout01 ALL=(ALL) NOPASSWD:ALL
```

Apéndice L

navegacion_autonoma_node.py (laptop o de escritorio)

```
#!/usr/bin/env python

# ---------- DEPENDENCIAS Y BIBLIOTECAS ----------
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import time
from time import time as Time_c
import random
import math

import rospy
import serial     # esta la usas solo si recibes los datos de la IMU directamente
from std_msgs.msg import Int32MultiArray
from std_msgs.msg import Float32
from geometry_msgs.msg import Vector3Stamped
from geometry_msgs.msg import Pose

i = 0
datos_angulos = np.genfromtxt('/home/sigmadrian/Curso_ROS/catkin_ws/src/Bipedo_CIA/
    scripts/Tray3.txt', delimiter = '\t')

dim_trayectoria = 6008    #5888 + 120
data = np.zeros((dim_trayectoria,12)) #trayectoria
a = 120

for n in xrange(0, dim_trayectoria - a):
    data[n+a,0] = datos_angulos[n,0]
```

```
28      data[n+a,1] = datos_angulos[n,1]
29      data[n+a,2] = datos_angulos[n,2]
30      data[n+a,3] = datos_angulos[n,3]
31      data[n+a,4] = datos_angulos[n,4]
32      data[n+a,5] = datos_angulos[n,5]
33      data[n+a,6] = datos_angulos[n,6]
34      data[n+a,7] = datos_angulos[n,7]
35      data[n+a,8] = datos_angulos[n,8]
36      data[n+a,9] = datos_angulos[n,9]
37      data[n+a,10] = datos_angulos[n,10]
38      data[n+a,11] = datos_angulos[n,11]
39
40  datos_recabados = np.zeros((dim_trayectoria,25))
41
42  eP_pitch = np.arange(-90, 90,0.1)
43
44  eP_pitch_ENG = fuzz.trimf(eP_pitch, [-90,-90,-15])#error negativo grande
45  eP_pitch_ENP = fuzz.trimf(eP_pitch,[-60,-15,0])#error negativo pequeno
46  eP_pitch_ECC = fuzz.trapmf(eP_pitch,[-15,-5,5,15])#error cercano cero
47  eP_pitch_EPP = fuzz.trimf(eP_pitch,[0,15,60])#error positivo pequeno
48  eP_pitch_EPG = fuzz.trimf(eP_pitch, [15,90,90])#error positivo grande
49
50  eI_pitch = np.arange(-200, 200,0.1)
51
52  eI_pitch_ENG = fuzz.trimf(eI_pitch, [-200,-200,-50])#error negativo grande
53  eI_pitch_ENP = fuzz.trimf(eI_pitch,[-100,-50,0])#error negativo pequeno
54  eI_pitch_ECC = fuzz.trimf(eI_pitch,[-50,0,50])#error cercano cero
55  eI_pitch_EPP = fuzz.trimf(eI_pitch,[0,50,100])#error positivo pequeno
56  eI_pitch_EPG = fuzz.trimf(eI_pitch, [50,200,200])#error positivo grande
57
58  eD_pitch = np.arange(-300, 300,0.1)
59
60  eD_pitch_ENG = fuzz.trimf(eD_pitch, [-300,-300,-50])#error negativo grande
61  eD_pitch_ENP = fuzz.trimf(eD_pitch,[-100,-50,0])#error negativo pequeno
62  eD_pitch_ECC = fuzz.trapmf(eD_pitch,[-50,-5,5,50])#error cercano cero
63  eD_pitch_EPP = fuzz.trimf(eD_pitch,[0,50,100])#error positivo pequeno
64  eD_pitch_EPG = fuzz.trimf(eD_pitch, [50,300,300])#error positivo grande
65
66  # eP_roll = np.arange(-90, 90,0.1)
67
68  # eP_roll_ENG = fuzz.trimf(eP_roll, [-90,-90,-15])#error negativo grande
69  # eP_roll_ENP = fuzz.trimf(eP_roll,[-60,-25,0])#error negativo pequeno
70  # eP_roll_ECC = fuzz.trapmf(eP_roll,[-25,-5,5,25])#error cercano cero
71  # eP_roll_EPP = fuzz.trimf(eP_roll,[0,25,60])#error positivo pequeno
72  # eP_roll_EPG = fuzz.trimf(eP_roll, [15,90,90])#error positivo grande
```

```
73
74 KP = np.arange(0,1.5,0.1)#angulo inicial robot: 115
75
76 KP_mf1 = fuzz.trapmf(KP, [0, 0,0.2,0.4])#angulo cercano cero
77 KP_mf2 = fuzz.trimf(KP, [0.2, 0.4, 0.6])#angulo pequeno cero
78 KP_mf3 = fuzz.trimf(KP,[0.4,0.6,0.8])#angulo mitad
79 KP_mf4 = fuzz.trimf(KP, [0.6, 0.9, 1.1])#angulo pequeno mitad
80 KP_mf5 = fuzz.trapmf(KP, [0.9, 1.1, 1.5,1.5])#angulo casi final
81
82
83 KI = np.arange(0,1.5,0.1)#angulo inicial robot: 65
84
85 KI_mf1 = fuzz.trapmf(KI, [0, 0,0.2,0.4])#angulo cercano cero
86 KI_mf2 = fuzz.trimf(KI, [0.2, 0.4, 0.6])#angulo pequeno cero
87 KI_mf3 = fuzz.trimf(KI,[0.4,0.6,0.8])#angulo mitad
88 KI_mf4 = fuzz.trimf(KI, [0.6, 0.9, 1.1])#angulo pequeno mitad
89 KI_mf5 = fuzz.trapmf(KI, [0.9, 1.1, 1.5,1.5])#angulo casi final
90
91 KD = np.arange(0,0.5,0.1)#angulo inicial robot: 140
92
93 KD_mf1 = fuzz.trimf(KD,[ 0,0,0.1])#angulo cercano cero
94 KD_mf2 = fuzz.trimf(KD, [0, 0.1, 0.2])#angulo pequeno cero
95 KD_mf3 = fuzz.trimf(KD,[0.1,0.2,0.3])#angulo mitad
96 KD_mf4 = fuzz.trimf(KD, [0.2, 0.3, 0.4])#angulo pequeno mitad
97 KD_mf5 = fuzz.trapmf(KD, [0.3, 0.4, 0.5,0.5])#angulo casi final
98
99 KP_roll = np.arange(0,1.5,0.1)#angulo inicial robot: 115
100
101 KP_roll_mf1 = fuzz.trapmf(KP_roll, [0, 0,0.1,0.2])#angulo cercano cero
102 KP_roll_mf2 = fuzz.trimf(KP_roll, [0.1, 0.2, 0.4])#angulo pequeno cero
103 KP_roll_mf3 = fuzz.trimf(KP_roll,[0.2,0.4,0.8])#angulo mitad
104 KP_roll_mf4 = fuzz.trimf(KP_roll, [0.6, 0.8, 1.1])#angulo pequeno mitad
105 KP_roll_mf5 = fuzz.trapmf(KP_roll, [0.8, 1.1, 1.5,1.5])#angulo casi final
106
107 KI_roll = np.arange(0,1.5,0.1)#angulo inicial robot: 65
108
109 KI_roll_mf1 = fuzz.trapmf(KI, [0, 0,0.2,0.4])#angulo cercano cero
110 KI_roll_mf2 = fuzz.trimf(KI, [0.2, 0.4, 0.6])#angulo pequeno cero
111 KI_roll_mf3 = fuzz.trimf(KI,[0.4,0.6,0.8])#angulo mitad
112 KI_roll_mf4 = fuzz.trimf(KI, [0.6, 0.9, 1.1])#angulo pequeno mitad
113 KI_roll_mf5 = fuzz.trapmf(KI, [0.9, 1.1, 1.5,1.5])#angulo casi final
114
115 KD_roll = np.arange(0,0.5,0.1)#angulo inicial robot: 140
116
117 KD_roll_mf1 = fuzz.trimf(KD,[ 0,0,0.05])#angulo cercano cero
```

```
KD_roll_mf2 = fuzz.trimf(KD, [0.05, 0.1, 0.2])#angulo pequeno cero
KD_roll_mf3 = fuzz.trimf(KD,[0.1,0.2,0.3])#angulo mitad
KD_roll_mf4 = fuzz.trimf(KD, [0.2, 0.3, 0.4])#angulo pequeno mitad
KD_roll_mf5 = fuzz.trapmf(KD, [0.3, 0.4, 0.5,0.5])#angulo casi final

Ang_RodillaD = np.arange(0,50,1)#angulo inicial robot: 140

Ang_RodillaD_mf1 = fuzz.trapmf(Ang_RodillaD,[0,0,5,10])#angulo cercano cero
Ang_RodillaD_mf2 = fuzz.trimf(Ang_RodillaD,[5, 10,15])#angulo pequeno cero
Ang_RodillaD_mf3 = fuzz.trimf(Ang_RodillaD,[5,15,25])#angulo mitad
Ang_RodillaD_mf4 = fuzz.trimf(Ang_RodillaD,[15,30,35])#angulo pequeno mitad
Ang_RodillaD_mf5 = fuzz.trapmf(Ang_RodillaD,[30,35,50,50])#angulo casi final

Ang_RodillaI = np.arange(0,50,1)#angulo inicial robot: 140

Ang_RodillaI_mf1 = fuzz.trapmf(Ang_RodillaI,[0,0,5,10])#angulo cercano cero
Ang_RodillaI_mf2 = fuzz.trimf(Ang_RodillaI,[5, 10,15])#angulo pequeno cero
Ang_RodillaI_mf3 = fuzz.trimf(Ang_RodillaI,[5,15,25])#angulo mitad
Ang_RodillaI_mf4 = fuzz.trimf(Ang_RodillaI,[15,30,35])#angulo pequeno mitad
Ang_RodillaI_mf5 = fuzz.trapmf(Ang_RodillaI,[30,35,50,50])#angulo casi final

def ErrorP_category(error_in):
    eP_pitch_ENG_cat = fuzz.interp_membership(eP_pitch,eP_pitch_ENG,error_in)
    eP_pitch_ENP_cat = fuzz.interp_membership(eP_pitch,eP_pitch_ENP,error_in)
    eP_pitch_ECC_cat = fuzz.interp_membership(eP_pitch,eP_pitch_ECC,error_in)
    eP_pitch_EPP_cat = fuzz.interp_membership(eP_pitch,eP_pitch_EPP,error_in)
    eP_pitch_EPG_cat = fuzz.interp_membership(eP_pitch,eP_pitch_EPG,error_in)
    return dict(NG = eP_pitch_ENG_cat, NP = eP_pitch_ENP_cat, CE = eP_pitch_ECC_cat
    , PP = eP_pitch_EPP_cat, PG = eP_pitch_EPG_cat)

def ErrorI_category(error_in):
    eI_pitch_ENG_cat = fuzz.interp_membership(eI_pitch,eI_pitch_ENG,error_in)
    eI_pitch_ENP_cat = fuzz.interp_membership(eI_pitch,eI_pitch_ENP,error_in)
    eI_pitch_ECC_cat = fuzz.interp_membership(eI_pitch,eI_pitch_ECC,error_in)
    eI_pitch_EPP_cat = fuzz.interp_membership(eI_pitch,eI_pitch_EPP,error_in)
    eI_pitch_EPG_cat = fuzz.interp_membership(eI_pitch,eI_pitch_EPG,error_in)
    return dict(NG = eI_pitch_ENG_cat, NP = eI_pitch_ENP_cat, CE = eI_pitch_ECC_cat
    , PP = eI_pitch_EPP_cat, PG = eI_pitch_EPG_cat)

def ErrorD_category(error_in):
    eD_pitch_ENG_cat = fuzz.interp_membership(eD_pitch,eD_pitch_ENG,error_in)
    eD_pitch_ENP_cat = fuzz.interp_membership(eD_pitch,eD_pitch_ENP,error_in)
    eD_pitch_ECC_cat = fuzz.interp_membership(eD_pitch,eD_pitch_ECC,error_in)
    eD_pitch_EPP_cat = fuzz.interp_membership(eD_pitch,eD_pitch_EPP,error_in)
    eD_pitch_EPG_cat = fuzz.interp_membership(eD_pitch,eD_pitch_EPG,error_in)
```

```
161     return dict(NG = eD_pitch_ENG_cat, NP = eD_pitch_ENP_cat, CE = eD_pitch_ECC_cat
        , PP = eD_pitch_EPP_cat, PG = eD_pitch_EPG_cat)
162 
163 # def ErrorP_roll_category(error_in):
164 #     eP_roll_ENG_cat = fuzz.interp_membership(eP_roll,eP_roll_ENG,error_in)
165 #     eP_roll_ENP_cat = fuzz.interp_membership(eP_roll,eP_roll_ENP,error_in)
166 #     eP_roll_ECC_cat = fuzz.interp_membership(eP_roll,eP_roll_ECC,error_in)
167 #     eP_roll_EPP_cat = fuzz.interp_membership(eP_roll,eP_roll_EPP,error_in)
168 #     eP_roll_EPG_cat = fuzz.interp_membership(eP_roll,eP_roll_EPG,error_in)
169 #     return dict(NG = eP_roll_ENG_cat, NP = eP_roll_ENP_cat, CE = eP_roll_ECC_cat,
        PP = eP_roll_EPP_cat, PG = eP_roll_EPG_cat)
170 
171 
172 def marcador(tray):
173     marcador = 0
174 
175     if tray > 500:#control postura
176         marcador = 1
177 
178     if tray > 640:#inicio trayectoria
179         marcador = 2
180 
181     if tray > 1440: #caminata 1
182         marcador = 3
183 
184     if tray > 2240:#caminata 2
185         marcador = 4
186 
187     if tray > 3040:#caminata 3
188         marcador = 5
189 
190     if tray > 3180:#fin trayectoria
191         marcador = 6
192 
193     if tray > 3680:#control postura
194         marcador = 7
195 
196     return marcador
197 
198 def Limites(angulo):
199     if angulo > 180:
200         angulo = 180
201     if angulo < 0:
202         angulo = 0
203     return angulo
```

```
204
205 # ---------------------------- VARIABLES GLOBALES ----------------------------
206 PWM = [0,0,0,0,0,0,0,0,0,0,0,0] # 12 Servomotores
207 RPY = [0.0,0.0,0.0]      # Arreglo donde se leen Roll, Pitch & Yaw
208 offset_frente = 0.0       # Offset de calibracion hacia el frente del robot
209 bandera_calibracion = False
210 bandera_giro_derecho = False
211 bandera_giro_izquierdo = False
212 bandera_linea_recta = False
213 yaw_objetivo = 10.0         # Angulo final al que se quiere llegar
214 angulo_giro = 10
215
216 ahora_t = 0
217 ultimo_t = 0
218 dt = 0
219 error_sum_pitch = 0
220 error_sum_roll = 0
221 contador = 0
222 tiempo = 0
223 suma_tiempo = 0
224 filter_rate_pitch = 0
225 filter_rate_roll = 0
226
227
228 # ================== CALLBACK de ROS ==============================
229 def CallBack_RPY(RPY_data):
230     # Uso de variables globales
231     global i,ahora_t, ultimo_t, dt, error_sum_pitch, error_sum_roll, contador,
    tiempo, suma_tiempo, filter_rate_pitch, filter_rate_roll
232
233     ahora_t = Time_c() # Para comenzar a medir tiempo
234     # Almacenamiento de datos de la IMU
235     RPY[0] = RPY_data.vector.x*57.2958
236     RPY[1] = RPY_data.vector.y*57.2958
237
238     # Calculo de los errores
239     error_roll = 0 - RPY[0]
240     error_pitch = 0 - RPY[1]
241     error_rate_pitch = 0
242     error_rate_roll = 0
243     filter_rate_pitch = filter_rate_pitch + 0.1*(error_rate_pitch -
    filter_rate_pitch)
244     filter_rate_roll = filter_rate_roll +0.1*(error_rate_roll - filter_rate_roll)
245     ultimo_t = Time_c() # Para terminar de medir tiempo
246     dt = ultimo_t - ahora_t # Diferencial de tiempo
```

```
247     suma_tiempo  += dt
248     error_sum_pitch += (error_pitch*dt)
249     error_sum_roll += (error_roll*dt)
250 
251     #categorias errores pitch
252     eP_rule = ErrorP_category(error_pitch)
253     eI_rule = ErrorI_category(error_sum_pitch)
254     eD_rule = ErrorD_category(filter_rate_pitch)
255 
256     #categorias error roll
257     eP_roll_rule = ErrorP_category(error_roll)
258     eI_roll_rule = ErrorI_category(error_sum_roll)
259     eD_roll_rule = ErrorD_category(filter_rate_roll)
260 
261     #ERROR PITCH
262     regla1_KP = np.fmax(eP_rule['NG'],eD_rule['NG'])
263     regla2_KP = np.fmax(eP_rule['NP'],eD_rule['NP'])
264     regla3_KP = np.fmax(eP_rule['CE'],eD_rule['CE'])
265     regla4_KP = np.fmax(eP_rule['PP'],eD_rule['PP'])
266     regla5_KP = np.fmax(eP_rule['PG'],eD_rule['PG'])
267 
268     #ERROR SUMA PITCH
269     regla1_KI = np.fmax(eP_rule['NG'],eI_rule['NG'])
270     regla2_KI = np.fmax(eP_rule['NP'],eI_rule['NP'])
271     regla3_KI = np.fmax(eP_rule['CE'],eI_rule['CE'])
272     regla4_KI = np.fmax(eP_rule['PP'],eI_rule['PP'])
273     regla5_KI = np.fmax(eP_rule['PG'],eI_rule['PG'])
274 
275     #CAMBIO ERROR  PITCH
276     regla1_KD = np.fmax(eP_rule['NG'],eD_rule['NG'])
277     regla2_KD = eP_rule['NP']
278     regla3_KD = eP_rule['CE']
279     regla4_KD = eP_rule['PP']
280     regla5_KD = np.fmax(eP_rule['PG'],eD_rule['PG'])
281 
282     #ERROR ROLL
283     regla1_KP_roll = np.fmax(eP_roll_rule['NG'],eD_roll_rule['NG'])
284     regla2_KP_roll = np.fmax(eP_roll_rule['NP'],eD_roll_rule['NP'])
285     regla3_KP_roll = np.fmax(eP_roll_rule['CE'],eD_roll_rule['CE'])
286     regla4_KP_roll = np.fmax(eP_roll_rule['PP'],eD_roll_rule['PP'])
287     regla5_KP_roll = np.fmax(eP_roll_rule['PG'],eD_roll_rule['PG'])
288 
289     #ERROR SUMA ROLL
290     regla1_KI_roll = np.fmax(eP_roll_rule['NG'],eI_roll_rule['NG'])
291     regla2_KI_roll = np.fmax(eP_roll_rule['NP'],eI_roll_rule['NP'])
```

```
292     regla3_KI_roll = np.fmax(eP_roll_rule['CE'],eI_roll_rule['CE'])
293     regla4_KI_roll = np.fmax(eP_roll_rule['PP'],eI_roll_rule['PP'])
294     regla5_KI_roll = np.fmax(eP_roll_rule['PG'],eI_roll_rule['PG'])
295 
296     #CAMBIO ERROR  ROLL
297     regla1_KD_roll = np.fmax(eP_roll_rule['NG'],eD_roll_rule['NG'])
298     regla2_KD_roll = eP_roll_rule['NP']
299     regla3_KD_roll = eP_roll_rule['CE']
300     regla4_KD_roll = eP_roll_rule['PP']
301     regla5_KD_roll = np.fmax(eP_roll_rule['PG'],eD_roll_rule['PG'])
302 
303     # #ERROR ROLL cadera
304     # regla1_KP_roll_c = eP_roll_rule['NG']
305     # regla2_KP_roll_c = eP_roll_rule['NP']
306     # regla3_KP_roll_c = eP_roll_rule['CE']
307     # regla4_KP_roll_c = eP_roll_rule['PP']
308     # regla5_KP_roll_c = eP_roll_rule['PG']
309 
310     #ANGULO RODILLA
311     regla1_ang_rodillaD = eP_roll_rule['NG']
312     regla2_ang_rodillaD = eP_roll_rule['NP']
313     regla3_ang_rodillaD = eP_roll_rule['CE']
314     regla4_ang_rodillaD = eP_roll_rule['PP']
315     regla5_ang_rodillaD = eP_roll_rule['PG']
316 
317     #ANGULO RODILLA
318     regla1_ang_rodillaI = eP_roll_rule['NG']
319     regla2_ang_rodillaI = eP_roll_rule['NP']
320     regla3_ang_rodillaI = eP_roll_rule['CE']
321     regla4_ang_rodillaI = eP_roll_rule['PP']
322     regla5_ang_rodillaI = eP_roll_rule['PG']
323 
324     #Implementacion de reglas KP,KI,KD
325     regla1_KP_act = np.fmin(regla1_KP,KP_mf5)#si ENG y  entonces KP mf5 nivel alto
326     regla2_KP_act = np.fmin(regla2_KP,KP_mf3)#si ENP y entonces KP mf2, nivel bajo
327     regla3_KP_act = np.fmin(regla3_KP,KP_mf1)#si ECC y entonces KP mf3
328     regla4_KP_act = np.fmin(regla4_KP,KP_mf3)#si EPP y entonces KP mf4
329     regla5_KP_act = np.fmin(regla5_KP,KP_mf5)#si EPG y entonces KP mf5
330 
331     regla1_KI_act = np.fmin(regla1_KI,KI_mf1)#si ENG y ENG entonces KI mf1
332     regla2_KI_act = np.fmin(regla2_KI,KI_mf3)#si ENP y entonces KI mf2
333     regla3_KI_act = np.fmin(regla3_KI,KI_mf4)#si ECC y entonces KI mf3
334     regla4_KI_act = np.fmin(regla4_KI,KI_mf3)#si EPP y entonces KI mf4
335     regla5_KI_act = np.fmin(regla5_KI,KI_mf1)#si EPG y entonces KI mf5
336 
```

```
337     regla1_KD_act = np.fmin(regla1_KD,KD_mf4)#si ENG y ENG entonces KD mf1
338     regla2_KD_act = np.fmin(regla2_KD,KD_mf2)#si ENP y entonces KD mf2
339     regla3_KD_act = np.fmin(regla3_KD,KD_mf1)#si ECC y entonces KD mf3
340     regla4_KD_act = np.fmin(regla4_KD,KD_mf2)#si EPP y entonces KD mf4
341     regla5_KD_act = np.fmin(regla5_KD,KD_mf4)#si EPG y entonces KD mf5
342
343     regla1_KP_roll_act = np.fmin(regla1_KP_roll,KP_roll_mf2)#si ENG y  entonces KP
    mf5 nivel alto
344     regla2_KP_roll_act = np.fmin(regla2_KP_roll,KP_roll_mf1)#si ENP y entonces KP
    mf2, nivel bajo
345     regla3_KP_roll_act = np.fmin(regla3_KP_roll,KP_roll_mf1)#si ECC y entonces KP
    mf3
346     regla4_KP_roll_act = np.fmin(regla4_KP_roll,KP_roll_mf1)#si EPP y entonces KP
    mf4
347     regla5_KP_roll_act = np.fmin(regla5_KP_roll,KP_roll_mf2)#si EPG y entonces KP
    mf5
348
349     regla1_KI_roll_act = np.fmin(regla1_KI_roll,KI_roll_mf1)#si ENG y  entonces KP
    mf5 nivel alto
350     regla2_KI_roll_act = np.fmin(regla2_KI_roll,KI_roll_mf2)#si ENP y entonces KP
    mf2, nivel bajo
351     regla3_KI_roll_act = np.fmin(regla3_KI_roll,KI_roll_mf3)#si ECC y entonces KP
    mf3
352     regla4_KI_roll_act = np.fmin(regla4_KI_roll,KI_roll_mf2)#si EPP y entonces KP
    mf4
353     regla5_KI_roll_act = np.fmin(regla5_KI_roll,KI_roll_mf1)#si EPG y entonces KP
    mf5
354
355     regla1_KD_roll_act = np.fmin(regla1_KD_roll,KD_roll_mf2)#si ENG y  entonces KP
    mf5 nivel alto
356     regla2_KD_roll_act = np.fmin(regla2_KD_roll,KD_roll_mf1)#si ENP y entonces KP
    mf2, nivel bajo
357     regla3_KD_roll_act = np.fmin(regla3_KD_roll,KD_roll_mf1)#si ECC y entonces KP
    mf3
358     regla4_KD_roll_act = np.fmin(regla4_KD_roll,KD_roll_mf1)#si EPP y entonces KP
    mf4
359     regla5_KD_roll_act = np.fmin(regla5_KD_roll,KD_roll_mf2)#si EPG y entonces KP
    mf5
360
361     regla1_ang_rodillaD_act = np.fmin(regla1_ang_rodillaD,Ang_RodillaD_mf5)#si ENG
    y ENG entonces KD mf1
362     regla2_ang_rodillaD_act = np.fmin(regla2_ang_rodillaD,Ang_RodillaD_mf3)#si ENP
    y entonces KD mf2
363     regla3_ang_rodillaD_act = np.fmin(regla3_ang_rodillaD,Ang_RodillaD_mf1)#si ECC
    y entonces KD mf3
```

```
364     regla4_ang_rodillaD_act = np.fmin(regla4_ang_rodillaD,Ang_RodillaD_mf3)#si EPP
    y entonces KD mf4
365     regla5_ang_rodillaD_act = np.fmin(regla5_ang_rodillaD,Ang_RodillaD_mf5)#si EPG
    y entonces KD mf5
366
367     regla1_ang_rodillaI_act = np.fmin(regla1_ang_rodillaI,Ang_RodillaI_mf5)#si ENG
    y ENG entonces KD mf1
368     regla2_ang_rodillaI_act = np.fmin(regla2_ang_rodillaI,Ang_RodillaI_mf3)#si ENP
    y entonces KD mf2
369     regla3_ang_rodillaI_act = np.fmin(regla3_ang_rodillaI,Ang_RodillaI_mf1)#si ECC
    y entonces KD mf3
370     regla4_ang_rodillaI_act = np.fmin(regla4_ang_rodillaI,Ang_RodillaI_mf3)#si EPP
    y entonces KD mf4
371     regla5_ang_rodillaI_act = np.fmin(regla5_ang_rodillaI,Ang_RodillaI_mf5)#si EPG
    y entonces KD mf5
372
373     SumaMembresiasKP = np.fmax(regla1_KP_act,np.fmax(regla2_KP_act,np.fmax(
    regla3_KP_act,np.fmax(regla4_KP_act,regla5_KP_act))))
374     SumaMembresiasKI =  np.fmax(regla1_KI_act,np.fmax(regla2_KI_act,np.fmax(
    regla3_KI_act,np.fmax(regla4_KI_act,regla5_KI_act))))
375     SumaMembresiasKD = np.fmax(regla1_KD_act,np.fmax(regla2_KD_act,np.fmax(
    regla3_KD_act,np.fmax(regla4_KD_act,regla5_KD_act))))
376
377     SumaMembresiasKP_roll = np.fmax(regla1_KP_roll_act,np.fmax(regla2_KP_roll_act,
    np.fmax(regla3_KP_roll_act,np.fmax(regla4_KP_roll_act,regla5_KP_roll_act))))
378     SumaMembresiasKI_roll = np.fmax(regla1_KI_roll_act,np.fmax(regla2_KI_roll_act,
    np.fmax(regla3_KI_roll_act,np.fmax(regla4_KI_roll_act,regla5_KI_roll_act))))
379     SumaMembresiasKD_roll = np.fmax(regla1_KD_roll_act,np.fmax(regla2_KD_roll_act,
    np.fmax(regla3_KD_roll_act,np.fmax(regla4_KD_roll_act,regla5_KD_roll_act))))
380
381     SumaMembresias_a_D = np.fmax(regla1_ang_rodillaD_act,np.fmax(
    regla2_ang_rodillaD_act,np.fmax(regla3_ang_rodillaD_act,np.fmax(
    regla4_ang_rodillaD_act,regla5_ang_rodillaD_act))))
382     SumaMembresias_a_I = np.fmax(regla1_ang_rodillaI_act,np.fmax(
    regla2_ang_rodillaI_act,np.fmax(regla3_ang_rodillaI_act,np.fmax(
    regla4_ang_rodillaI_act,regla5_ang_rodillaI_act))))
383
384     #defuzzification, resultado:
385     Kp_pitch = fuzz.centroid(KP, SumaMembresiasKP)
386     Ki_pitch = fuzz.centroid(KI, SumaMembresiasKI)
387     Kd_pitch = fuzz.centroid(KD, SumaMembresiasKD)
388
389     Kp_roll = fuzz.centroid(KP_roll, SumaMembresiasKP_roll)
390     Ki_roll = fuzz.centroid(KI_roll, SumaMembresiasKI_roll)
391     Kd_roll = fuzz.centroid(KD_roll, SumaMembresiasKD_roll)
```

```
392
393     a_D = fuzz.centroid(Ang_RodillaD, SumaMembresias_a_D)
394     a_I = fuzz.centroid(Ang_RodillaI, SumaMembresias_a_I)
395
396     #ADQUISiCION DE DATOS, GRAFICAS
397     '''datos_recabados[i,0] = suma_tiempo
398     datos_recabados[i,1] = error_pitch
399     datos_recabados[i,2] = error_roll
400     datos_recabados[i,3] = error_rate_pitch
401     datos_recabados[i,4] = error_rate_roll
402     datos_recabados[i,5] = filter_rate_pitch
403     datos_recabados[i,6] = filter_rate_roll
404     datos_recabados[i,7] = error_sum_pitch
405     datos_recabados[i,8] = error_sum_roll
406     datos_recabados[i,9] = Kp_pitch
407     datos_recabados[i,10] = Ki_pitch
408     datos_recabados[i,11] = Kd_pitch
409     datos_recabados[i,12] = Kp_roll
410     datos_recabados[i,13] = Ki_roll
411     datos_recabados[i,14] = Kd_roll'''
412
413     if (i < a + 1):
414         servo11 = 90
415         servo12 = 90
416         servo21 = Limites(int(90 + Kp_roll*error_roll + Ki_roll*error_sum_roll +
    Kd_roll*error_rate_roll))
417         servo22 = Limites(int(90 + Kp_roll*error_roll + Ki_roll*error_sum_roll +
    Kd_roll*error_rate_roll))
418         servo31 = Limites(int(150 - Kp_pitch*error_pitch - Ki_pitch*error_sum_pitch
     - Kd_pitch*error_rate_pitch - a_D))#cadera PID
419         servo32 = Limites(int(45 + Kp_pitch*error_pitch + Ki_pitch*error_sum_pitch
    + Kd_pitch*error_rate_pitch +  a_I))#cadera PID
420         servo41 = Limites(int(65 - Kp_pitch*error_pitch - Ki_pitch*error_sum_pitch
    - Kd_pitch*error_rate_pitch - a_D))#Rodilla
421         servo42 = Limites(int(120 + Kp_pitch*error_pitch + Ki_pitch*error_sum_pitch
     + Kd_pitch*error_rate_pitch + a_I))#Rodilla
422         servo51 = Limites(int(120 - Kp_pitch*error_pitch - Ki_pitch*error_sum_pitch
     - Kd_pitch*error_rate_pitch - a_D))#Tobillo PID
423         servo52 = Limites(int(65 + Kp_pitch*error_pitch + Ki_pitch*error_sum_pitch
    + Kd_pitch*error_rate_pitch + a_I))#Tobillo PID
424         servo61 = Limites(int(90 + Kp_roll*error_roll+ Ki_roll*error_sum_roll +
    Kd_roll*error_rate_roll))
425         servo62 = Limites(int(95 + Kp_roll*error_roll+ Ki_roll*error_sum_roll +
    Kd_roll*error_rate_roll))
426     if(i > a ):
```

```
427         servo12 = int(data[i,0])
428         servo11 = int(data[i,1])
429         servo22 = Limites(int(data[i,2] + Kp_roll*error_roll + Ki_roll*
    error_sum_roll + Kd_roll*error_rate_roll))
430         servo21 = Limites(int(data[i,3] + Kp_roll*error_roll + Ki_roll*
    error_sum_roll + Kd_roll*error_rate_roll))
431         servo32 = Limites(int(data[i,4] + Kp_pitch*error_pitch + Ki_pitch*
    error_sum_pitch + Kd_pitch*error_rate_pitch ))#+  a_I))#cadera PID
432         servo31 = Limites(int(data[i,5] - Kp_pitch*error_pitch - Ki_pitch*
    error_sum_pitch - Kd_pitch*error_rate_pitch ))#- a_D))#cadera PID
433         servo42 = Limites(int(data[i,6] + Kp_pitch*error_pitch + Ki_pitch*
    error_sum_pitch + Kd_pitch*error_rate_pitch ))#+ a_I))#Rodilla
434         servo41 = Limites(int(data[i,7] - Kp_pitch*error_pitch - Ki_pitch*
    error_sum_pitch - Kd_pitch*error_rate_pitch ))#- a_D))#Rodilla
435         servo52 = Limites(int(data[i,8] + Kp_pitch*error_pitch + Ki_pitch*
    error_sum_pitch + Kd_pitch*error_rate_pitch ))#+ a_I))#Tobillo PID
436         servo51 = Limites(int(data[i,9] - Kp_pitch*error_pitch - Ki_pitch*
    error_sum_pitch - Kd_pitch*error_rate_pitch ))#- a_D))#Tobillo PID
437         servo62 = Limites(int(data[i,10] + Kp_roll*error_roll+ Ki_roll*
    error_sum_roll + Kd_roll*error_rate_roll))
438         servo61 = Limites(int(data[i,11] + Kp_roll*error_roll+ Ki_roll*
    error_sum_roll + Kd_roll*error_rate_roll))
439
440     PWM[0] = servo11             # servo11
441     PWM[1] = servo12             # servo12
442     PWM[2] = servo21             # servo21
443     PWM[3] = servo22             # servo22
444     PWM[4] = servo31#cadera PID  # servo31
445     PWM[5] = servo32#cadera PID  # servo32
446     PWM[6] = servo41#Rodilla     # servo41
447     PWM[7] = servo42#Rodilla     # servo42
448     PWM[8] = servo51#Tobillo PID # servo51
449     PWM[9] = servo52-5#Tobillo PID # servo52
450     PWM[10] = servo61-10          # servo61
451     PWM[11] = servo62-5          # servo62
452
453     i = i + 1 # Para ser utilizado por las matrices de arriba
454
455 # ================== INTERRUPCION POR /yaw Y LEY DE CONTROL DE NAVEGACION
    ==================
456 def CallBack_Yaw(Yaw_data):      # La MPU-6050 entrega angulos de 0 a 360 grados
457     global bandera_calibracion, offset_frente
458
459     if bandera_calibracion == False:
460         offset_frente = 300.0 - Yaw_data.data
```

```
461
462         bandera_calibracion = True
463         #print "offset_frente = ",offset_frente
464
465     else:
466         RPY[2] = Yaw_data.data + offset_frente
467         #print "Yaw_actual = ",RPY[2]," Yaw_objetivo= ",yaw_objetivo
468
469 # ========================= Publicador y Subscriptor ========================
470 def Bipedo_Publisher_and_Subscriber():
471     global bandera_giro_izquierdo, bandera_giro_derecho, bandera_linea_recta
472
473     # Nombre del nodo, false para que no despliegue numero aleatorio
474     rospy.init_node('navegacion_autonoma_node', anonymous=False)
475     print("Inicializando navegacion_autonoma_node...")
476
477     # Definicion del publicador
478     pub = rospy.Publisher('/servos_topic', Int32MultiArray, queue_size=10)
479     #pub_plot = rospy.Publisher('/servo_plot_compu', Pose, queue_size=10)
480
481     # Definicion del subscriptor
482     rospy.Subscriber('/imu/rpy', Vector3Stamped, CallBack_RPY)
483     rospy.Subscriber('/yaw', Float32, CallBack_Yaw)
484     #rospy.spin() <-------- este es remplazado por while not rospy.is_shutdown()
485
486     # Velocidad del programa
487     rate = rospy.Rate(500) # (10) = 10[Hz]
488
489     # Arreglo de 12 servos donde se almacenaran los datos a publicar
490     angulos = Int32MultiArray()
491     #angulo_plot = Pose()
492
493     # Datos iniciales
494     angulos_iniciales = [90,90,90,87,141,57,60,120,120,60,87,87]
495     angulos.data = angulos_iniciales
496     #angulo_plot.position.x = 0
497
498     print "El robot marchara hasta encontrar el angulo de giro programado"
499     print "El angulo objetivo es: ",yaw_objetivo
500
501     # Para indicarle al robot que girara hacia la izquierda o derecha
502     if  True:#yaw_objetivo > 90 and yaw_objetivo < 270:
503         bandera_giro_derecho = False
504         bandera_giro_izquierdo = True
505         print "El giro sera izquierdo"
```

```
506     #elif yaw_objetivo < 90 or yaw_objetivo > 270:
507     #    bandera_giro_derecho = True
508     #    bandera_giro_izquierdo = False
509     #    print "El giro sera derecho"
510     #sleep(2)
511 
512     # ============== WHILE LOOP ============================
513     while not rospy.is_shutdown():  # Confirma que todo esta bien
514 
515         # Uso de variables globales
516         global i,PWM, error_roll, error_pitch, error_sum, filter_rate, Kp, Ki,
    datos_recabados
517 
518         # Almacenamiento en arreglo del publicador <------------ CAMBIAR A FORLOOP
519 
520         # Filtro para evitar que de golpes de senal abruptos:
521         if RPY[0] == 0 and RPY[1] == 0:
522             angulos.data = [90,90,90,87,141,57,60,120,120,60,87,87];
523         else:
524             # Con giro hacia la derecha
525             if bandera_giro_derecho == True and bandera_giro_izquierdo == False:
526                 angulos.data[0] = PWM[0] - angulo_giro           # Cambiar solo el
    valor del angulo de giro y su denominador
527                 angulos.data[1] = PWM[1] - angulo_giro -(angulos_iniciales[0] -PWM[1])
    *(angulo_giro/4)
528 
529             # Con giro hacia la izquierda
530             elif bandera_giro_derecho == False and bandera_giro_izquierdo == True:
531                 angulos.data[0] = PWM[0]+angulo_giro+(PWM[0]-angulos_iniciales[0])*(
    angulo_giro/4)
532                 angulos.data[1] = PWM[1]+angulo_giro
533 
534             angulos.data[2] = PWM[2]
535             angulos.data[3] = PWM[3]
536             angulos.data[4] = PWM[4]
537             angulos.data[5] = PWM[5]
538             angulos.data[6] = PWM[6]
539             angulos.data[7] = PWM[7]
540             angulos.data[8] = PWM[8]
541             angulos.data[9] = PWM[9]
542             angulos.data[10] = PWM[10]
543             angulos.data[11] = PWM[11]
544 
545             #angulo_plot.position.x = angulos.data[10] # el servo de posicion 7 de
    muestra
```

```
546
547         # Impresion de datos de calculo
548         #print "eR: ",error_roll," eP: ", error_pitch," eI: ",error_sum_pitch," eD:
    ",filter_rate_pitch, " Kp_pitch: ",Kp_pitch, " Ki_pitch: ",Ki_pitch,"contador",i
    #, " Marcador: ",marcador(i)
549         #print "Roll: ", RPY[0], " Pitch: ", RPY[1]
550
551         # Publicacion del mensaje
552         pub.publish(angulos)
553         #pub_plot.publish(angulo_plot)
554         # Delay del programa definido en el rate
555         rate.sleep()
556
557         print "",RPY[2]
558         error_Yaw = abs(yaw_objetivo-RPY[2])
559
560         tolerancia_servos = abs(90 - angulos.data[10])
561
562         if error_Yaw < 3:
563           bandera_llegada = True
564         else:
565           bandera_llegada = False
566
567         if bandera_giro_derecho == True:
568           print "Al robot le faltan ",error_Yaw," angulos a la derecha para llegar.
    "
569         else:
570           print "Al robot le faltan ",error_Yaw," angulos a la izquierda para
    llegar."
571
572         # LLEGADA
573         if bandera_llegada == True:
574             print "SE HA LLEGADO AL ANGULO OBJETIVO, TRAYECTORIA TERMINADA."
575             exit()
576
577         #Sustituye el for xrange del antiguo control_completo, para que pueda
    salirse
578         # del programa una vez completado la dimension de la trayectoria
579         if i >= dim_trayectoria:
580           print "TRAYECTORIA TERMINADA"
581         exit()
582
583 # ================ LOOP ====================
584 if __name__ == '__main__':
585     try:
```

```
586         Bipedo_Publisher_and_Subscriber()
587     except rospy.ROSInterruptException:
588         pass
```

Apéndice M

Repositorios

Nota: Recordar que los paquetes que se descarguen van dentro de la carpeta ***src*** *del espacio de trabajo (usualmente llamado* ***catkin_ws****).*

Archivo *Aprendiendo_ROS.pdf* junto con el espacio de trabajo como ejemplo en el repositorio de GitHub:

https://github.com/AdrianSiGmA/Aprendiendo-ROS.git

Paquete de ROS Bipedo_CIA (equipo externo) en el repositorio de GitHub:

https://github.com/AdrianSiGmA/Bipedo_CIA.git

Paquete de ROS Bipedo (Raspberry) en el repositorio de GitHub:

https://github.com/AdrianSiGmA/Bipedo.git

La imagen funcional de **16GB de la Raspberry con ROS** se encuentra en la carpeta de */home/Respaldo Robot_Bipedo/IMG_16GB_ROS.img* de la computadora de escritorio del CIA.

Apéndice N

Comandos más usados en el trabajo

Linux - Ubuntu

Abrir una terminal: oprimir las teclas **Ctrl + Alt + T** al mismo tiempo

Terminar un proceso: oprimir las teclas **Ctrl + C** al mismo tiempo dentro de la terminal

Salvar cualquier programa (recomendado cada 15 minutos): oprimir las teclas **Ctrl + S** al mismo tiempo

Buscar algún comando previamente usado en la terminal: oprimir las teclas **Ctrl + R** al mismo tiempo

ROS

Crear un paquete: **catkin_create_pkg** + *DEPENDENCIAS* (como **rospy, roscpp, std_msgs, geometry_msgs, etc.**)

Correr un nodo: **rosrun** *NOMBRE_PAQUETE NOMBRE_NODO*

Lista de tópicos: **rostopic list**

Ver un tópico: **rostopic echo** *NOMBRE_TÓPICO*

Apéndice Ñ

Advertencias y posibles errores con solución

Jerarquización de Nodos: Hay que tener cuidado con el uso de Tópicos y el levantamiento de Nodos, ya que cada uno tiene una jerarquía de operación al ser requerimientos de otro. Por ejemplo, si se ejecuta cualquier Nodo de **control** sin antes haber iniciado el Nodo de la **tiva** (servomotores) y posteriormente se intenta levantar este último, el robot puede llegar a sobresaltarse motrizmente el robot, causando posibles daños en los cables o una caída al suelo.

Solución: Analizar la jerarquización antes de lanzar el **roslaunch** o cualquier Nodo independiente. Por ejemplo, si se sabe que el control de postura o marcha bípeda funciona con valores de PWM de entre **50°** y **180°** del Tópico **/servos_topic**, y que el valor de **0°** puede ocasionar escrituras en PWM fatales para el robot, evitar a toda costa que se escriban dichos ángulos erróneos. **¿Cómo hacerlo?**, con condicionantes. Si por alguna razón el Nodo de la **um7** deja de transmitir información, éste proporcionará múltiples datos de **0°**. Entonces se debe avisar al Nodo de **control** que publique datos **hasta que las inclinaciones del Tópico /imu/rpy** sean diferentes a **0°**.

Otra solución igual de directa sería modificar el Nodo de la **tiva**. Si por alguna razón se reciben datos de PWM del Tópico **/servos_topic** iguales a **0°**, simplemente no escribir los datos a los servomotores **hasta que sean diferentes a 0°**.

Afortunadamente, ya se tiene implementado este pequeño filtro de datos en los Nodos de **control_postura_node.py** y **control_completo_node.py** que pueden verse en los Apéndices **A** y **B** respectivamente, pero se explica la clave de filtrado para futuros controladores con ROS.

Publicación al mismo Tópico: Si se ejecutaran hipotéticamente varios Nodos, por ejemplo: **control_postura_node.py**, **control_completo_node.py** y **navegacion_autonoma_node.py**, lo único que sucedería es que el Bípedo se confundiría con los datos a la hora de escribirlos a los actuadores, comportándose de una manera totalmente incongruente a sus cálculos.

Solución: En ROS es recomendable aprovechar los Tópicos al máximo y suscribirse cuantas veces se requiera. Lo que no es recomendable es publicar al mismo Tópico con más de un Nodo a la vez. **La solución es evitar correr múltiples Nodos que publiquen al mismo Tópico**.

Energia IDE como superusuario: Algunas veces, la IDE puede presentar problemas de carga de código a la tarjeta debido a una mala instalación de controladores.

Solución: Siempre es mejor utilizar la instalación de la página oficial [19] porque aminora los posibles errores. Sin embargo, en Ubuntu 16.04 hay que tener perfectamente ubicada la carpeta de sketches y la carpeta de instalación que se haya elegido de la IDE. Ante todo, se le debe dar permisos de superusuario a la ejecución del programa:
sudo ./*UBICACIÓN***/energia**

Identificador de puertos ttyACMX: Esto es muy frecuente en Ubuntu. A diferencia de Windows 10, no se le asigna un identificador único a los dispositivos que se conectan. Generalmente se desconectan y se vuelven a conectar, ocasionando que ROS intente ejecutar un Nodo para el módulo incorrecto, dejando incompleto el proceso de levantamiento de Nodos y Tópicos con el **roslaunch**.

Solución 1: En el mejor de los casos, uno se puede conectar por teclado USB a la Raspberry y ver en un monitor o pantalla lo que sucedió probando los puertos. Ejecutar todos los Nodos manualmente casi siempre funciona, pero para eso sirve el **roslaunch**, para que futuros desarrolladores no pierdan tiempo ejecutando procesos uno por uno.

Solución 2: **Existe una forma de obtener el identificador único de un dispositivo, en especial de los microcontroladores**. Este concepto fue aplicado en la Sección **7** de identificadores automáticos de puertos.

Puertos visibles, pero inactivos: Suele suceder que a pesar de haberle dado permisos a los puertos con el comando:
sudo chmod 777 /dev/ttyACM*X*
y lograrlos visualizar sin ningún problema con **ls /dev**, aún así no funcionen con ROS.

Solución: Lo más recomendable en estos casos es **desconectar y conectar el módulo**. En casos un poco más extremos, hay que desenergizar totalmente el robot, teniendo cuidado en apagar la Raspberry de manera correcta antes de hacerlo con **sudo shutdown -h now**.

Visualización de Tópicos desde otro equipo: Si ya se levantó el ROS MASTER en la Raspberry y aún así no se pueden ver los Tópicos publicados en el equipo externo, quiere decir que existe una incongruencia entre las especificaciones de direcciones IP entre ambos. La misma **solución** a este problema analiza las propiedades del */.bashrc* en la Sección **5.5** de configuración de las IP.

Ejecución del roscore sin el robot bípedo: Se debe recordar que ROS funcionará correctamente si se le especifica en dónde está ubicado el ROS MASTER.
Solución: Sin embargo, si se desean realizar pruebas sin el robot bípedo (únicamente en el equipo externo), se deben quitar las especificaciones de dirección IP del ROS MASTER en el mismo archivo */.bashrc*. Con comentarlas es suficiente.

***Wrong checksum error* en la ejecución de Nodos:** Como ya se mencionó en la Sección **8.2.2** de la especulación acerca de evitar tener un publicador y un suscriptor en el mismo microcontrolador, pueden llegar a presentarse retrasos, o incluso desincronizaciones entre el ROS MASTER y el equipo externo.

Solución: El publicador del microcontrolador sólo se utilizó para fines de graficación. Por un periodo corto de tiempo el robot funciona sin ningún problema, pero después de ciertos segundos existe la posibilidad de fallos de comunicación. **Se deben quitar (comentar) en los códigos los publicadores innecesarios para el funcionamiento del Bípedo**.

Nota: Procurar darle permisos de escritura a los scripts de Python (Nodos) y ejecutar como superusuario todas las operaciones referentes al manejo de puertos.

Todos estos fueron algunos de los problemas con los que se enfrentó el desarrollo de esta tesis. Hay decenas más de ellos que no se mencionaron y que pueden solucionarse analizando el problema desde el fundamento teórico. Algunos otros, por medio de prueba y error. Otros, revisando los foros en la comunidad de programadores. Y otros, simplemente es mejor verlos desde una nueva perspectiva y comenzar a buscar alternativas para evitarlos.