XXXVIII Jornadas de Automática
https://doi.org/10.17979/spudc.9788497497749.0686

Estructura de control en ROS y modos de marcha basados en máquinas de estados de un robot hexápodo

Raúl Cebolla Arroyo
raul.cebolla.arroyo@alumnos.upm.es

Jorge De León Rivas, Antonio Barrientos
CAR UPM-CSIC, jorge.deleon, antonio.barrientos@upm.es

Resumen

Los sistemas de control en ROS basados en el paquete ros_control requieren la coordinación y publicación de cantidades ingentes de información. En los siguientes apartados se describe la estructura y herramientas que simplifican el control al uso de funciones simples en C++. Estas se recogen como métodos de una clase que simboliza el actuador de un robot hexápodo de exploración. Los modos de marcha de este robot, se planifican mediante máquinas de estados que simplifican su planteamiento y aumentan la capacidad de respuesta ante un fallo.

Palabras clave: ROS, interfaz, máquinas de estado.

1. Introducción

El robot hexápodo que es tratado en este caso es una versión del RHex desarrollado por el grupo de Robotica y Cibernetica de la Universidad Politecnica de Madrid [6] [8] [10] , este robot fue concebido por proyectos de diversas universidades estadounidenses [6] como robot de búsqueda y rescate [7]. La particularidad de este robot reside en la forma en C de sus patas, posee tres de estas patas por cada lado articuladas a una base.

La configuración del robot le dota de una capacidad única para adoptar multitud de diferentes modos de marcha [4] [9] [5]. Esto hace que el control de cada una de las patas que lo componen sea lo más preciso posible. De igual forma se necesita que el modo de controlarlo sea de fácil accesibilidad al encargado de programar los diferentes modos de marcha.

El sistema de control del robot se ha desarrollado sobre ROS haciendo uso del paquete *ros_control*. Este tipo de sistemas de control generan multitud de diferentes mensajes y servicios sobre los que hay que actuar, causando que ejecutar una acción simple se vuelva ciertamente caótico.

Teniendo esta última idea en mente, se ha desarrollado un paquete de ROS capaz de simplificar el tráfico de información a manejar. El paquete permite que el control del robot pueda hacerse mediante llamadas a métodos de una clase que simbolice cada una de las patas en C.

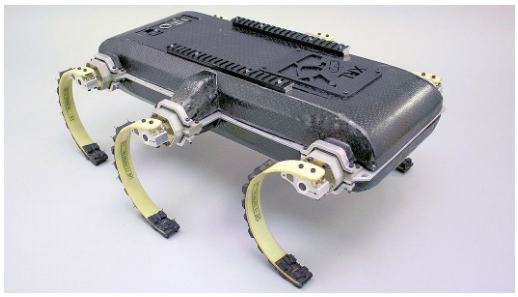

Figura 1: Robot RHex. Fuente: [3]

Esta simplificación resulta fundamental a la hora de programar los modos de marcha. Estos se implementan mediante máquinas de estado. Siguiendo este método, se mejora la legibilidad y planteamiento de las diferentes marchas, de la misma forma se mejora la flexibilidad de las mismas ya que sólo se tiene que incluir un nuevo estado y modificar las transiciones pertinentes. Con esto se obtiene que el modo de marcha pueda implementar respuestas ante fallo de distinto tipo y paradas de emergencia.

La flexibilidad que aportan las máquinas de estado hace que sea idóneo para un robot cuyos modos de marcha son muy diferentes entre sí y que cada día aparecen nuevos para el mismo robot.

En el estudio del estado del arte no se han encontrado robots cuyos modos de marchas se implementen mediante máquinas de estado. Por tanto, este trabajo supone una incursión en una nueva idea que puede llevar a nuevas percepciones sobre cómo se implementan modos de marcha.

2. Descripción del robot

El robot sobre el que vamos a desarrollar el trabajo pertenece al robcib. Robot Hexápodo desarrollado por el grupo de Robotica y Cibernetica de la Universidad Politecnica de Madrid [6] [8] cuya configuración y especificaciones se orientan hacia tareas de búsqueda y rescate.

La principal característica del RHex son sus patas en C. Sus seis patas se disponen de forma simétrica respecto a la dirección de avance normal del robot. Estas patas disponen de cierta flexibilidad [8][2], de esta forma el robot puede aprovechar el efecto de ballesta que crean para avanzar a mayor velocidad o incluso realizar saltos.

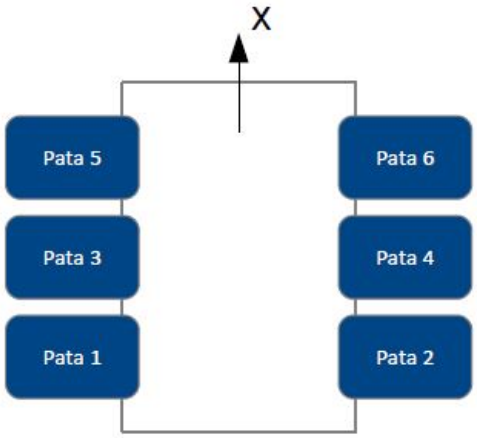

Figura 2: Numeración de las patas.

Para planteamientos futuros, se tomará la numeración de las patas que se expone en la figura 2 como convenio.

Con estas características, el hexápodo puede adaptar multitud de diferentes modos de marcha. El robot puede desempeñar desde pequeños saltos, subir y bajar escaleras [4] [1], avanzar de forma cuadrúpeda o incluso bípeda [5].

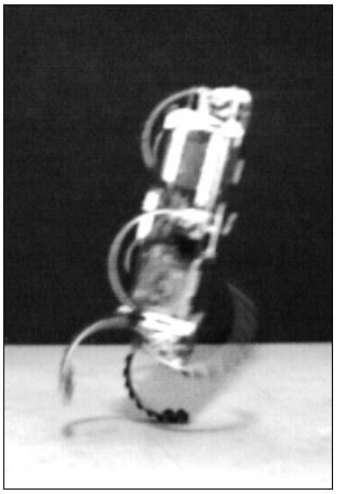

Figura 3: Marcha bípeda del RHex. Fuente: [5].

Se demuestra de forma indudable la versatilidad del robot en cuestión, los diferentes modos de marcha no es sino una prueba fehaciente de ello. Aún quedan por desarrollarse marchas que ni se han planteado, por eso es crucial que en un modelo de investigación se pueda acceder a la programación de las mismas de una manera sencilla y sobre todo, sin necesidad de poseer conocimientos sobre la arquitectura de control a bajo nivel.

La capacidad para superar terrenos accidentados del robot es difícilmente alcanzable por otros, esto lo hace idóneo para trabajos de búsqueda y rescate. Por tanto, es de esperar que el robot se encuentre con situaciones no previstas en su programación. Se necesita de un recurso que haga del robot, un sistema robusto ante este tipo de fallos. Este se encuentra en las máquinas de estado.

Una máquina de estado no es más que un planteamiento dónde el robot realiza transiciones entre estados, en cada estado el robot se comanda de una manera distinta, por ejemplo patas 1, 4, 5 a fase aérea mientras que el resto realizan la fase terrestre.

Al programar una máquina de estados se puede establecer una ruta de transiciones que per-

mitan al robot reaccionar ante imprevistos, de tal forma que nunca se encuentre bloqueado. Además, resulta sencillo reprogramar los diferentes modos de marcha, ya que esta metodología es muy flexible a la hora de incluir un nuevo estado.

3. Estructura de control

El sistema de control se ha planteado sobre el sistema operativo de robots más extendido, ROS, por tanto, para entender cómo se ha simplificado el comandado de cada una de las patas, es necesario saber la estructura de control que se ha planteado sobre el robot.

El sistema de control se ha basado sobre el paquete *ros_control*. Este paquete estructura el sistema de control mediante dos elementos: las interfaces de hardware y los controladores.

Las interfaces de hardware sirven como conexión a ROS de los elementos reales, bien sean sensores o actuadores, mediante estos se conoce dónde se posiciona en memoria la información de los sensores y se puede ejercer sobre los actuadores.

Por otro lado, los controladores asumen las mismas funciones y tareas que un regulador o controlador de la teoría de control. En este caso, los controladores de *ros_control* son de tipo PID.

Teniendo en cuenta las características del robot, es de especial interés que en cualquier momento durante la ejecución de un modo de marcha se pueda cambiar de control de posición a control de velocidad en tiempo real. *Ros_control* permite cambiar de tipo de controlador mediante un agente que recibe el nombre de *controller_manager*, este permite cargar, eliminar o cambiar controladores en tiempo real, más adelante se estudiará cuáles son los servicios y mensajes a los que se debe acudir.

Es crucial saber que sobre un solo actuador, se puede tener activo un único controlador. Sin embargo, puede tener más controladores pausados que no actúen sobre él.

En base a lo establecido en estos últimos párrafos, cada una de las patas tendrá asociados dos controladores:

- Controlador de posición
- Controlador de velocidad

Que, respectivamente, recibirán el nombre de:

- *pata_position_controller*
- *pata_velocity_controller*

Los tipos de controladores que incluye *ros_control* por defecto son suficientes para las necesidades del robot por lo que no se necesita crear uno nuevo. De estos controladores se tomarán los que se basan en el esfuerzo, *effort_controller*, y dentro de estos de posición, *JointPositionController*, y de velocidad, *JointVelocityController*.

Para inicializar cada uno de los controladores es necesario una serie de parámetros, estos parámetros, tanto para el controlador de posición como el de velocidad son:

- El nombre de la unión que está controlada.
- Los valores de las constantes proporcional (P), derivativa (D) e integral (I).

Los valores de cada una de las constantes han sido ajustados mediante métodos eurísticos. Observando el comportamiento de las patas con distintos valores se ha determinado lo siguiente:

- Con acción proporcional el intervalo de establecimiento es menor de 1 segundo en el aire, suficiente para los requisitos del robot, por tanto no requiere de acción derivativa para alcanzar las especificaciones dinámicas.
- La acción derivativa, hace aumentar mucho la sobreoscilación, descartándose totalmente su uso.
- Se ha añadido una acción integral mínima para asegurar que el sistema sea robusto ante perturbaciones de bloqueo. Especialmente útil cuando la pata necesita alzar el robot desde el suelo.

Siguiendo estas conclusiones y mediante procedimientos eurísticos, los valores de cada una de las constantes en ambos casos es el siguiente:

$$Parametros\ PID \begin{cases} P &= 10 \\ I &= 0{,}2 \\ D &= 0 \end{cases} \quad (1)$$

Estos parámetros deben ser cargados en el servidor de parámetros de ROS. Para poder inicializar los controladores se pude hacer una llamada al nodo */controller_spawner* pasando como argumento el nombre de los controladores a cargar.

Para conocer el estado[1] de los actuadores se puede incluir otro controlador, el */jointStateController*, también estándar de *ros_control* que emite el mensaje de topic */robcib_hexapod/joint_state* con el estado de las patas. Esto será necesario para la planificación de modos de marcha en ocasiones donde, por ejemplo, la condición de la transición sea el paso por una posición concreta.

Una vez que se ha desarrollado cómo se estructuran los controladores y que el robot puede ser actuado, necesitamos saber cómo realizar ese comandado.

En primera instancia hay que especificar cómo se gestionan los controladores mediante el *controller_manager*. Este agente se ejecuta dentro del nodo principal del robot, donde previamente se ha establecido las interfaces de hardware mecionadas anteriormente, con esto se ofrece una serie de servicios con los que se le puede hacer peticiones sobre la carga, descarga y cambio de controladores:

- ***/robcib_hexapod/controller_manager /list_controllers***: Devuelve una lista de los controladores y su estado (ejecutando o parado).
- ***/robcib_hexapod/controller_manager /load_controller***: carga un controlador.
- ***/robcib_hexapod/controller_manager /unload_controller***: para y quita un controlador.
- ***/robcib_hexapod/controller_manager /switch_controller***: cambia el controlador de una interfaz de hardware. Para un actuador y dos controladores cargados del mismo, cambia el activo al que se desee.

Por cada controlador instanciado en ROS, se tienen una serie de mensajes y servicios con los cuales se puede gobernar sobre el controlador, se estudiarán dos aspectos: el cambio de los parámetros PID y la orden de referencia.

[1] En este caso, estado de un actuador se corresponde a la posición, velocidad y esfuerzo en unidades del sistema internacional.

Para cambiar los parámetros de control durante la ejecución, se debe recurrir a los siguientes servicios dependiendo del controlador activo en ese momento:

- ***/robcib_hexapod/pata1_position_controller /set_parameters***
- ***/robcib_hexapod/pata1_velocity_controller /set_parameters***

Los campos a completar se corresponden con los parámetros PID de igual manera que se ha realizado previamente.

Por otro lado, para ordenar una posición o velocidad de referencia que debe alcanzar como entrada escalón, se recurre a la publicación de los siguientes mensajes, una vez de nuevo, dependiendo del controlador activo:

- ***/robcib_hexapod/pata1_position_controller /command***
- ***/robcib_hexapod/pata1_velocity_controller /command***

Para que el sistema no falle, este último mensaje debe ser publicado de forma periódica.

Todos los ejemplos han sido empleando la pata 1, pero son perfectamente extrapolables a cualquier otra pata cambiando el número por su correspondiente.

Es necesario aclarar, que cada vez que se cambia o se para un controlador, este pierde la referencia de posición o velocidad y por tanto, es necesario que se esté publicando el mensaje de comandado frecuentemente.

4. Interfaz de control

En este punto el robot es capaz de ser controlado, pero la cantidad de información, mensajes y servicios es abrumadora. Para simplificar este problema se ha introducido una nueva capa sobre el control, que manejará este tráfico.

El paquete resultante fue */robcib_hexapod_controller_interface* que implementa una interfaz sobre el control que lo simplifica.

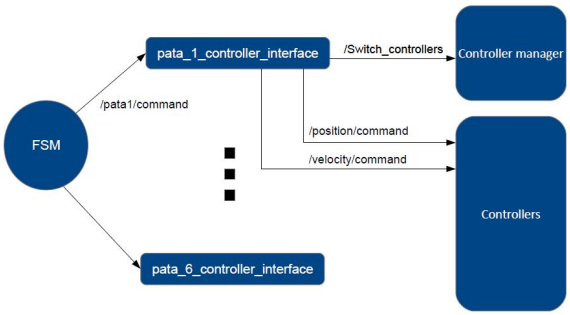

Figura 4: Diagrama de funcionamiento del */robcib_hexapod_controller_interface*.

El paquete desarrollado, contiene un nodo de ROS con el mismo nombre que publica todos los mensajes necesarios y llama a los servicios pertinentes, según lo que se le haya pedido en el servicio creado de forma específica para la funcionalidad.

Los principales aspectos que se pretenden conseguir con este paquete son:

1. Simplificar el control del robot.
2. Asegurar que todos los mensajes necesarios de control queden cubiertos.
3. Prevenir fallos antes de comandar.
4. Publicar los mensajes de comando de forma periódica.
5. Simplificar el código.

Siguiendo estas metas, el nodo realiza las siguientes funciones:

- Toma en cuenta qué tipo de control está activo, cambia mediante el servicio */switch_controller* el controlador que es necesario.
- Comprueba que el valor del comando es coherente y publica de forma periódica el mensaje */command* del tipo que se necesite, bien sea de posición o de velocidad.

Se debe ejecutar un nodo por cada una de las patas. Esto es muy importante ya que para que se publique la información de manera correcta el nodo debe saber qué pata está controlando, esto se consigue pasando como argumento la pata correspondiente, por ejemplo, para la pata 1 se pasaría: pata1.

Para superar el obstáculo que pueda suponer que el mensaje de comando se tenga que emitir cada cierto tiempo, el nodo de *robcib_hexapod_controller_interface* se encargará de ello. El mensaje de */command* es publicado de forma periódica, en este caso con una frecuencia de 10 Hz. Dos motivos han llevado a esta decisión:

- Frecuencias muy altas suponen mucho coste computacional.
- Frecuencias por debajo de los 2 Hz se ha comprobado que el comportamiento es errático.

Teniendo estos dos factores en cuenta se ha decidido con eurística que la frecuencia de mensaje debe ser de los 10 Hz mencionados anteriormente.

El nodo creado debe ser comandado a su vez de alguna forma, para esto se ha escogido un servicio creado ad-hoc para la aplicación. Este servicio se convierte en el único que el usuario debe tener en cuenta para el control.

Este servicio recibe el nombre de */c_leg_command*. Este, necesita dos argumentos:

- El valor de la nueva referencia en unidades del sistema internacional, esto es [rad] para posición y [rad/s] para velocidad.
- Si es valor de posición o de velocidad.

El servidor devuelve el mensaje de servicio con un campo booleano que indica si la instrucción ha podido realizarse (valor verdadero) o no (valor falso).

La cantidad de informacion a manejar se ha reducido significativamente con la inclusión de la interfaz de control, sin embargo, para que resulte aún mas sencillo se ha decidido incluir una librería de C++ dentro del mismo paquete.

De esta manera nació la librería *CLeg* que implementa la clase del mismo nombre cuyo objetivo es el repetido en múltiples ocasiones ya en el texto, simplificar y mejorar la accesibilidad al usuario. La librería busca implementar una clase que sea más intuitiva de manejar por el usuario y que a su vez mejore la legibilidad del código.

Para declarar una instancia de la clase pata C o *CLeg*, se necesitan de dos argumentos:

- El número de pata
- El *NodeHandle* del nodo que hace uso de la librería.

El primer argumento busca saber a qué pata hace referencia. El node handle es necesario para avisar al maestro de ROS de que es un cliente, necesario para llevar al cabo la llamada al servicio */c_leg_command*.

Los métodos públicos que permiten los objetos de la clase son los siguientes:

- ***updateState()***: recoge el último mensaje de */robcib_hexapod/joint_state* y actualiza los campos del objeto de posición, velocidad y esfuerzo.
- ***getPos()***: actualiza el estado y devuelve la posición de la pata en [rad].
- ***getVel()***: actualiza el estado y devuelve la velocidad de la pata en [rad/s].
- ***getEff()***: actualiza el estado y devuelve el esfuerzo de la pata en [Nm].
- ***getEstControl()***: devuelve TRUE si el control es de posición y FALSE en caso contrario.
- ***setPosition(float value)***: comanda el valor *value* como posición. Comprueba si el control es ya de posición, si no lo cambia y comienza a publicar de forma periódica el mensaje de comando.
- ***setVelocity(float value)***: comanda el valor *value* como velocidad. Comprueba si el control es ya de velocidad, si no lo cambia y comienza a publicar de forma periódica el mensaje de comando.

Con esto se completa la descripción de las herramientas para el desarrollo de los modos de marcha en el robot.

5. Modos de marcha basados en máquinas de estados

Para desarrollar cómo se implementan mediante máquinas de estados un modo de marcha, se ejemplificará mediante el planteamiento para la marcha de trípode alterno.

El trípode alterno es un modo de marcha característicos donde dos trípodes, constituidos por 3 patas distintas cada uno, realizan un semiciclo de avance sobre el terreno con una apertura que denominaremos como ángulo de avance, mientras que la otra gira a mayor velocidad en el aire buscando ponerse en la posición inicial en tierra y alternaran sus posiciones girando siempre los dos trípodes en el mismo sentido. Esta constituye una de las marchas más básicas y características del robot.

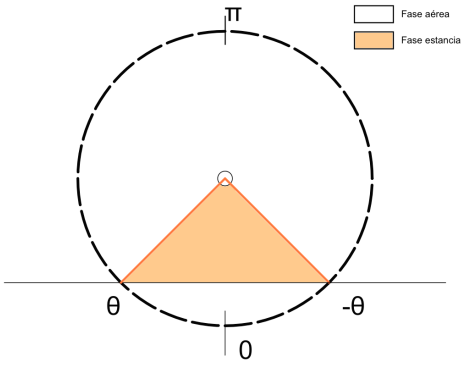

Figura 5: Fases aérea y terrestre.

En este caso, los trípodes 6 en cuestión serán:

- Trípode 1: patas 1 - 4 - 5
- Trípode 2: patas 2 - 3 - 6

Se parte de la premisa que el robot va a comenzar apoyado en tierra sobre su chasis.

Se distinguen tres estados en este modo de marcha:

- **Reposo:** Todas las patas en posición 0. El robot se queda erguido sobre sus patas.
- **Movimiento 1**: Tripode 1 en velocidad de tierra y el trípode 2 en velocidad de aire.
- **Movimiento 2:** Trípode 1 en velocidad de aire y trípode 2 en velocidad de tierra.

Una vez establecidos los estados, falta por determinar las condiciones de transición entre ellos:

- **Reposo a Mov 1**: las patas se encuentran en 0 y se da la señal de comienzo.

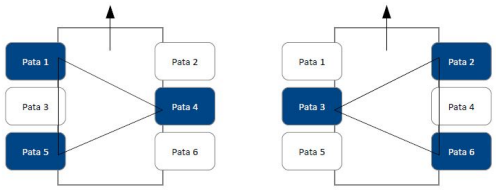

Figura 6: Trípodes.

- **Mov 1 a Mov 2**: trípode 1 en posición de despegue y trípode 2 en aterrizaje.
- **Mov 1 a Mov 2**: trípode 1 en posición de aterrizaje y trípode 2 en despegue.
- **De cualquier estado a Reposo**: señal de paro, emergencia o error no detectado.

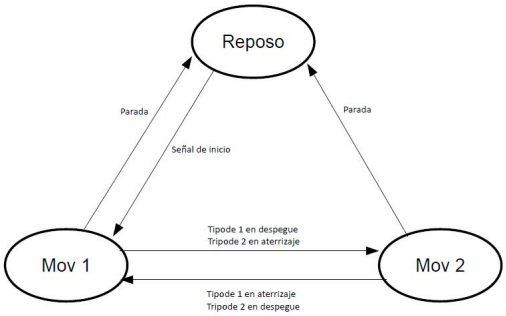

Figura 7: Máquina de estados del trípode alterno.

Una vez que se han definido los estados, las acciones en los estados y las transiciones entre los mismos, queda definida la máquina de estados de la marcha. La traducción a código en C++ con la librería *CLeg* es directa.

Es necesario distinguir que en el estado reposo las patas se comandan por posición, mientras que en el resto se hace por velocidad.

La metodología seguida es aplicable para cualquier modo de marcha. Además, dentro de este planteamiento se podría añadir un cuarto estado intermedio de posicionamiento de las patas a la primera posición de despegue y aterrizaje. Si se deseara realizar, basta con añadir un estado y sus transiciones correspondientes, siendo mínimos los cambios en el resto de elementos ya planteados.

Esto demuestra la flexibilidad del método para incluir nuevos estados como puedan ser los de respuesta ante perturbación o error causado por el entorno.

6. Conclusiones

En base a todos los puntos desarrollados, los robots hexápodos con un grado de libertad en sus articulaciones, tienen una gran versatilidad a la hora de superar diferentes terrenos y obstáculos debido a que puede adoptar una gran cantidad de marchas diferentes.

Esto causa que el control tenga cierta sofisticación y una gran cantidad de información que manejar. Esto se ha solucionado con el paquete *robcib_controller_interface* que implementa la clase *CLeg* cuyos métodos permiten el dar comandos a cada una de las patas, sin tener que publicar directamente los múltiples mensajes que nos requiere *ros_control*.

Los archivos y códigos fuente no se han publicado en un repositorio aún dado que se trata de un proyecto en desarrollo.

Por otro lado, los modos de marcha pueden ser muy complejos si se plantean como flujogramas, en cambio, mediante máquinas de estado se consigue que el problema sea modular y flexible facilitando su planteamiento. Las máquinas de estado admiten modificaciones con un menor número de cambios que otras metodologías como los flujogramas. Además, se puede añadir estados a los que pasar por defecto en caso de encontrarse en una situación no esperada, esto resulta en marchas más robustas ante errores o perturbaciones.

Agradecimientos

Esta investigación ha recibido fondos del proyecto RoboCity2030-III-CM (Robótica aplicada a la mejora de la calidad de vida de los ciudadanos. fase III; S2013/MIT-2748), financiado por los Programas de Actividades I+D en la Comunidad de Madrid y los Fondos Estructurales de la Unión Europea, y del proyecto DPI2014-56985-R (Protección robotizada de infraestructuras críticas), financiado por el Ministerio de Economía y Competitividad del Gobierno de España.

Referencias

[1] Campbell, D., and Buehler, M. Stair descent in the simple hexapod 'rhex'. *Ambulatory Robotics Laboratory, Centre for Intelligent Machines, McGill University* (s.f.).

[2] Jesus Tordesillas Torres, Jorge De Leon Rivas, J. D. C. A. B. Modelo cinematico de un robot hexapodo con c-legs. *ETSII - UPM* (2016).

[3] Kodlab. The rhex hexapedal robot. http://kodlab.seas.upenn.edu/RHex/Home. Consultado el 24 de Abril de 2017.

[4] Moore, E. Z. Leg design and stair climbing control for the rhex robotic hexapod. Master's thesis, Department of Mechanical Engineering McGill University, 2002.

[5] Neville, N., and Buehler, M. Towards bipedal running of a six-legged robot. *Ambulatory Robotics Laboratory Centre for Intelligent Machines, McGill University* (s.f.).

[6] Rivas, J. D. L. Definición y análisis de los modos de marcha de un robot hexápodo para tareas de búsqueda y rescate. Master's thesis, Escuela tecnica superior de ingenieros industriales - Universidad politecnica de Madrid, 2015.

[7] Siciliano, B., and Khatib, O., Eds. *Handbook of robotics*. Springer International Publishing, 2016.

[8] Torres, J. T. Diseño y simulación del sistema de locomoción de un robot hexápodo para tareas de búsqueda y rescate. Master's thesis, Escuela tecnica superior de ingenieros industriales - Universidad politecnica de Madrid, 2016.

[9] Uluc Saranli, A. A. R., and Koditschek, D. E. Model-based dynamic self-righting maneuvers for a hexapedal robot. *Robotics Institute, Carnegie Mellon University Department of Electrical Engineering and Computer Science The University of Michigan* (2004).

[10] Uluc Saranli, M. B., and Koditschek, D. E. Rhex - a simple and highly mobile hexapod robot. *Department of Electrical Engineering and Computer Science The University of Michigan and Center for Intelligent Machines McGill University Montreal* (2001).