Sincronización de Robots AIBO. Un Estudio Comparativo*

Diego Muñoz, Manel Velasco, Cecilio Angulo
Universitat Politècnica de Catalunya · UPC BarcelonaTech
Pau Gargallo 5. ESAII – Departament d'Automàtica, Barcelona, Spain 08028
diegoamg89@gmail.com, {manel.velasco, cecilio.angulo}@upc.edu

Abstract

El trabajo tiene por objetivo implementar un conjunto de módulos de software que permitan integrar la plataforma robótica AIBO en el marco de trabajo ROS (Robot Operating System). El problema principal radica en el método utilizado para comunicarlos. Inicialmente se estudiaron los diferentes software alternativos con los que poder operar. Luego se compararon varios métodos de programación, de forma que un análisis cuantitativo permitió decidir el método más adecuado. Finalmente se implementaron los módulos que facilitan la integración con ROS. Por un lado el módulo de control, al que se le han aplicado unos criterios de valoración y se ha puesto a prueba su funcionamiento, y por otro el módulo que facilita la integración de un modelo de visualización 3D. Con todo este trabajo desarrollado, se realizó la aplicación de sincronización de movimientos de un par de robots AIBO y se realizaron experimentaciones que muestran la bondad del trabajo desarrollado.

1. Introducción

Ocho años después de la retirada del mercado de la mascota robótica por excelencia, el robot AIBO, de la compañía japonesa Sony [Decuir *et al.*, 2004; Arevalillo-Herráez *et al.*, 2011], éstos permanecen encerrados en los armarios de muchos equipos de investigación, a pesar que en su día le sacaron un gran provecho tecnológico. Desde el Departamento de Automática de la Universitat Politècnica de Catalunya se ha lanzado una iniciativa global que tiene como objeto la recuperación de esta plataforma robótica para su uso en investigación y docencia. Mediante el uso de una comunicación inalámbrica, se facilitará el procesado de la carga de trabajo con mayores requerimientos en un procesador externo, mientras se continuan haciendo uso de las funcionalidades que hicieron de AIBO el producto robótico más sofisticado puesto en el mercado.

*Este trabajo ha sido financiado parcialmente por los proyectos de investigación PATRICIA (TIN2012-38416-C03-01) y EVENTS (DPI2010-18601), ambos financiados por el Ministerio de Economía y Competitividad del Gobierno de España.

El objetivo del presente estudio es realizar un demostrador del sistema en una tarea de sincronización de dos robots AIBO. Para llegar a esta meta es necesario ser capaz de adquirir los datos de las articulaciones y demás sensores de uno de los robots a la mayor velocidad posible, así como enviar los valores de control de las articulaciones al segundo AIBO. Para ello se han probado y analizado un conjunto de posibles lenguajes de programación / comunicación que permiten la intercomunicación del robot AIBO con ROS.

Con este fin, se han implementado una serie de experimentos que permitirán decidir qué método de adquisición de datos es más efectivo. La comparativa se ha basado en criterios como

- la frecuencia media de lectura, $\bar{x}$
- los intervalos de congelación en la lectura,
- la frecuencia máxima de envío,
- la capacidad de transmisión, y
- el retraso en el seguimiento de trayectorias.

Estos experimentos se han estructurado sobre un paradigma cliente servidor. En el que el cliente es el PC que contiene ROS y el servidor de datos es la plataforma AIBO.

AIBO soporta varios métodos para la extracción de datos hacia el PC. De entre ellos se ha elegido utilizar URBI [Baillie, 2005] y conexión directa utilizando telnet. Esta elección está fundamentada en la facilidad que presentan estos dos sistemas frente a métodos más complejos de comunicación.

Los métodos comparados en los experimentos han sido:

- Cliente implementado en C++ usando la librería liburbi.
- Dos clientes implementados en C++ usando la librería liburbi, una para la escritura y otro para la lectura.
- Comunicación usando telnet sobre un script de python.

Finalmente se ha optado por trabajar con URBI debido a que permite una comunicación sencilla entre el servidor, el AIBO, y un cliente externo.

Una vez alcanzado el anterior propósito y en aras de poner de manifiesto los resultados de los experimentos precedentes se realizaron algunos ejemplos de sincronización:

- Imitación de un robot AIBO a otro: esta aplicación requiere de una gran fluidez en el trafico de datos, convirtiéndose así en un excelente indicador del grado de capacidad de sincronización que es posible adquirir.

- Coordinación de movimientos para una tarea: este caso aportará una visión más útil de la sincronización de robots para realizar tareas.

Con el objetivo último de conseguir una comunicación entre robots AIBO de forma intuitiva a la vez que proporcionar una serie de herramientas muy útiles en los entornos de robótica móvil actuales, se usará ROS (Robot Opetating System) [Quigley *et al.*, 2009] como capa intermedia de la programación. Esta extensión facilitará la implementación de las aplicaciones anteriormente descritas, así como hacer uso de todas las librerías ya existentes en este entorno estándar de programación.

Por último, indicar que el trabajo aquí iniciado permite una rápida extensión hacia esquemas de robótica en la nube (*cloud robotics*) y de computación en red que son del todo interesantes de explorar.

2. El robot AIBO

La plataforma de estudio AIBO ha permitido la realización de numerosos proyectos de investigación, de índole muy diversa. Mayoritariamente se han llevado a cabo proyectos relacionados con visión artificial [Pérez *et al.*, 2010] y comportamientos en inteligencia artificial [Téllez *et al.*, 2006]. Un punto esencial en el desarrollo del robot AIBO fue que durante el período comprendido entre 1999 y 2008 fue la plataforma elegida para realizar la competición RoboCup Standard Plataform League, posteriormente sustituido por la plataforma NAO de Aldebaran. También en la prueba de simulacros de rescate, AIBO tuvo una gran acogida [Nardi and Lima, 2012]. La RoboCup fue fuente de inspiración para múltiples investigaciones que se pueden clasificar en tres grupos según su finalidad:

- Visión y reconocimiento de marcas para la posterior ubicación dentro del campo [Morales-Jiménez, 2007].
- Comunicaciones y control de los movimientos [Martínez Gómez, 2006].
- Definición de roles dentro del campo [Montero-Mora *et al.*, 2009].

Actualmente el robot AIBO ha caído en desuso, en parte por dejar de ser plataforma de la RoboCup y en parte porque dejó de ser mantenido por parte del fabricante. Las nuevas actualizaciones de los softwares que lo soportaban han dejado de ser compatibles, incluso aquellos que nacieron especialmente para el AIBO. Así es el caso de URBI, que desde la version 1.5, su librería *liburbi* no es compatible. Igualmente gran parte de la documentación de estos lenguajes ya no se encuentra en la red.

Han existido intentos de recuperar la plataforma AIBO, como es el caso de un proyecto que trataba de hacer compatible la version 2.3 de liburbi para AIBO [Kecsap, 2010]. Otro proyecto trataba de realizar la integración mediante EusLisp a ROS de varias plataformas, entre ellas AIBO. Sin embargo es muy escasa o ninguna la documentación referente a ello [Kertész, 2013].

3. Comparación de alternativas para la programación

Con la finalidad de implantar ROS en AIBO se plantea una primera elección del camino a seguir: (1) un núcleo de ROS embebido, es decir que sea en el propio AIBO donde se ejecute el proceso, (2) un núcleo de ROS en un ordenador remoto y comunicación entre ambos mediante una arquitectura cliente/servidor.

Para explorar la opción de un sistema embebido se ha buscado un punto de acceso al hardware con la intención de instalar una distribución linux. Sin embargo el único punto de acceso encontrado no respondía a ningún estándar, por lo que se desiste su utilización. Así, sólo es posible usar una arquitectura cliente/servidor con el módulo de ROS ejecutándose en el cliente y adquiriendo de forma sencilla y rápida el estado del AIBO y actuando sobre él.

Pese a que existen diferentes posibilidades de programación (OPEN-R, Tekkotsu, URBI), éste último tiene la ventaja de que no sería necesario desarrollar el servidor ya que el sistema operativo interactúa de forma remota. Además, el uso de URBI es relativamente sencillo, tanto desde el terminal como con el uso de la librería de C++. También existe bastante documentación. El único problema que se ha detectado es que la librería liburbi 1.5, la más reciente que es compatible con AIBO, no es compatible con un sistema linux de 64 bits.

Una vez escogido URBI como lenguaje de desarrollo, se plantean dos opciones:

- Usar liburbi con C++.
- Usar un script de python que use la terminal de URBI bajo una conexión telnet.

Para determinar la opción a elegir se realizaron una serie de experimentos comparativos sobre la eficacia de las comunicaciones de ambos métodos. Los experimentos se realizaron en ambas direcciones de la comunicación, por un lado la lectura y por otro el envío.

3.1. Lectura de datos

De los datos obtenidos de las 15 réplicas experimentales de cada uno de los dos métodos probados se han obtenido las diferencias de tiempos entre un dato y el siguiente y se traduce a frecuencia calculando el inverso.

Los resultados mostraron claramente que con el uso de liburbi se alcanzan frecuencias medias más altas ($\bar{x} = 13{,}611Hz$ y $\sigma = 13{,}271$ para liburbi y $\bar{x} = 3{,}000Hz$ y $\sigma = 2{,}763$ para python) en la mayoría de réplicas. Además, la mayoría de datos se envían más rápidos como se refleja en los valores de las medianas de liburbi que son mayores que los valores del tercer cuartil del metodo python. Por otro lado, los valores mínimos de cada método son muy similares y la variabilidad de los resultados de liburbi es mayor.

Tan importante como los valores medios de las frecuencias es comprobar si se producen bloqueos en la comunicación, es decir si durante un cierto período de tiempo no se actualiza la variable. Para verificar si se producen bloqueos es necesario atender a los valores mínimos de las series. Aparentemente liburbi presenta unos valores mínimos más bajos, sin embargo el análisis de la varianza (ANOVA) de los mínimos indica que

no se corresponden con poblaciones distintas, por lo que se acepta la hipótesis nula ($p\_valor = 0{,}4$ y $I.C. = 95\,\%$).

3.2. Envío de datos

El segundo experimento consiste en enviar una trayectoria sinusoidal punto a punto a una articulación y comprobar la respuesta. Para capturar los resultados se leerá la respuesta de la articulación usando los módulos de lectura del experimento anterior.

En este experimento se ha contado con 3 posibles opciones de programación. Esto es así debido a que la terminal de URBI usando un cliente telnet con python no permite la lectura y escritura simultánea, y en consecuencia se han tenido que desarrollar dos clientes telnet. Aunque si bien liburbi permite la lectura y escritura por un solo cliente, se ha considerado oportuno añadir un programa con dos clientes, equivalente al método utilizado con python.

En síntesis, los programas que se van a utilizar son los siguientes:

- Python con un cliente telnet para lectura y otro para escritura.
- C++ con un cliente URBI tanto para lectura como para escritura.
- C++ con un cliente URBI para lectura y otro para escritura.

El experimento se ha repetido cambiando la frecuencia a la que se envían los puntos de la trayectoria. Los valores de las frecuencias utilizados son: 1, 2, 5 y 10 Hz. Se han omitido mayores velocidades ya que el movimiento a una frecuencia de mas de 10 Hz era inconstante y discontinuo. Con la finalidad de obtener una mayor fiabilidad en los resultados se han realizado 3 réplicas por métodos y frecuencia. Se han descartado los experimentos en los que se ha producido un bloqueo.

De los resultados obtenidos se pueden distinguir dos tipos de errores que provocan que la trayectoria realizada no sea la correcta:

- La articulación no se mueve.
- No se recibe la posición actual. En este caso no se puede saber si la articulación se ha movido[1].

Se puede concluir que el seguimiento de la trayectoria es independiente de la frecuencia utilizada en este rango. Por otro lado también se observa que con un solo cliente liburbi se producen grandes bloqueos en la recepción y muestra una trayectoria más discontinua e inconstante. En lo que respecta a los otros dos métodos, dos clientes con liburbi y python, tienen un comportamiento aceptable y no se puede concluir que uno sea mejor que otro.

Para discernir cual de los dos métodos es más eficiente se han analizado las diferencias de tiempo entre el momento en que se envía la orden de los picos de la sinusoide y el momento en que se recibe que ha llegado a dicha posición. Para comprobar si existe diferencia entre las series se realiza una ANOVA sobre los datos. Con un intervalo de confianza del 95 % el p-valor es de 2,1e-7, por lo tanto no se acepta la hipótesis nula y se puede concluir que el retraso con C++ es significativamente menor.

En síntesis, el método liburbi se ha demostrado más eficaz en lo relativo a la recepción de datos, muestra un comportamiento similar a python en el seguimiento de las trayectorias con el uso de dos clientes y además se obtiene un retraso de seguimiento menor que python estadísticamente significativo. Adicionalmente, se ha de tener en consideración que el paquete será más complejo que los programas utilizados en los experimentos y dado que C++ se ejecuta a mayor velocidad que python las diferencias de ejecución podrían llegar a ser críticas. En consecuencia la librería liburbi usando dos clientes es el método elegido para implementar el paquete de ROS.

4. Implementación en ROS

ROS es el acrónimo de Robot Operating System, que lejos de ser un sistema operativo es más bien un marco de trabajo que proporciona unas herramientas y unas librerías para ayudar a desarrollar software para aplicaciones de robótica. Proporciona entre otras facilidades abstracciones de hardware, controladores para dispositivos, herramientas de visualización, comunicación por mensajes, administración de paquetes y todo ello bajo licencia de software libre. La principal característica sobre ROS es su sistema de comunicación. A bajo nivel está basado en el pase de mensajes que pueden ser leídos por diversos procesos simultáneamente. Dichos mensajes se escriben sobre unos canales de comunicación llamados *topics* a los que se puede acceder mediante métodos de publicación y subscripción. Sobre los tópicos se puede publicar o subscribir desde un terminal o bien cualquier objeto de ROS, denominados *nodes*. Todos los programas que se ejecutan sobre ROS deben ser creados dentro de un paquete (*package*).

4.1. Paquete *aibo_server*

Se pretende implementar un paquete que se conecte al robot AIBO dentro de una red local. Una vez conectado debe recoger los datos enviados por el servidor URBI del robot y publicarlos sobre una serie de tópicos. De forma inversa debe tratar los valores de las articulaciones del tópico al que está subscrito y enviarlos al servidor con el fin de actuar sobre la plataforma (ver Figura 1).

La implementación del paquete de ROS se realiza usando liburbi y dos clientes, uno para la recepción y otro para el envío. La estructura del programa implementado se puede ver en la Figura 2.

Por lo que respecta la adquisición de los valores de sensores y articulaciones, se han conseguido obtener trabajando a una frecuencia de 10Hz. Con objeto de verificar que el refresco de los tópicos es correcto se ha accedido a mostrar por el terminal todos ellos junto con su frecuencia de refresco.

Para realizar una valoración del envío de las posiciones se necesita de un paquete de ROS que permita cuantificar los resultados. El test aplicado es similar al realizado en las experimentaciones de la Sección 3. Se trata de enviar punto a punto una trayectoria sinusoidal a las posiciones de una articulación. Para la realización del test se ha implementado un sencillo programa que crea un nodo que publica sobre el tópi-

[1] Por inspección visual se ha comprobado que ambos errores son independientes.

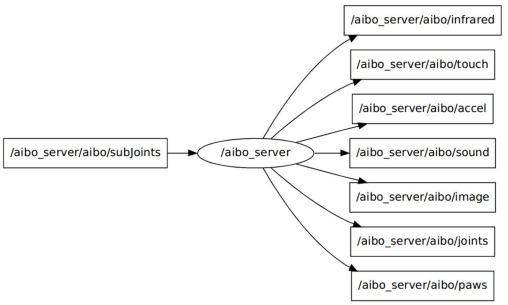

Figura 1: Nodo aibo_server.

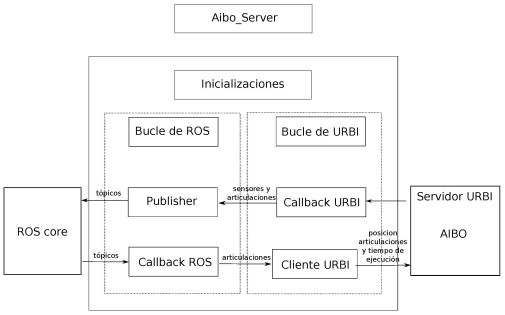

Figura 2: Estructura básica del ejecutable aibo_server.

co `/aibo_server/aibo/subjoints/jointRF1`. La estructura entre los nodos se muestra en la Figura 3.

De los resultados obtenidos en la realización de varias réplicas a diversas frecuencias se deduce que el seguimiento de la trayectoria lleva un retraso mínimo entorno a los 0.5 segundos, dato que coincide con el retraso medio obtenido en las experimentaciones de la subsección 3.2.

Por otro lado el modulo implantado no ha mejorado los resultados obtenidos en la subsección 3.2 en lo relativo a los bloqueos (Figura 4). Por lo tanto, cabe concluir que el módulo no proporciona las condiciones necesarias para aquellas aplicaciones en las que se requiere una lectura y una respuesta rápida del sistema.

4.2. Paquete *aibo_description*

Este paquete pretende ser el vínculo entre el robot AIBO y una de las herramientas más usadas de ROS para robots móviles, *rviz*[2], un visualizador 3D para aplicaciones de robótica que permite visualizar un modelo, el entorno sensado o un mapa virtual. El paquete consta de dos partes diferenciadas, la creación del modelo de AIBO para su representación en imagen 3D y la implementación de un nodo que sincronice el modelo con el robot.

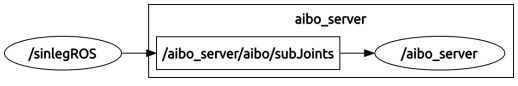

Figura 3: Estructura de ROS con los nodos aibo_server y SinLeg.

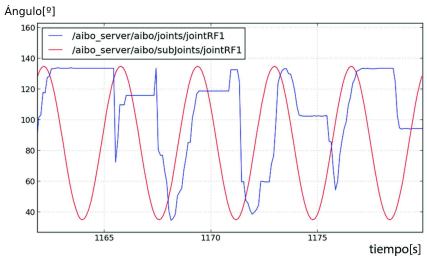

Figura 4: Entrada y respuesta del sistema ante una señal sinusoidal enviando puntos a 10Hz.

Para generar un modelo, se debe editar un fichero xml con formato URDF[3] (Unified Robots Description Format) donde se especifican las dimensiones del robot, las articulaciones y sus movimientos, la geometría de las extremidades y parámetros físicos como la masa o la inercia. Para el modelo de AIBO se han creado 22 partes y 21 articulaciones. Se han descartado en el modelo la boca y las orejas.

En cada una de las partes se debe definir la geometría y sus dimensiones. Con el fin de darle un aspecto visual más atractivo y mas semejante al robot AIBO real la geometría ha sido definida mediante archivos *stl* que contienen las figuras de cada una de las partes. A partir de un modelo sólido se han dividido y modificado algunas piezas usando los softwares de apoyo FreeCAD[4] y Netfabb[5]. El resultado desde la ventana de *rviz* se muestra en la Figura 5.

El nodo state_publisher está creado mediante un programa escrito en C++ (state_publisher.cpp) y está subscrito al tópico `/aibo_server/aibo/joints`, donde se encuentran las posiciones de las articulaciones del robot real, y las publica en un tópico llamado `joint_states` al que accede *rviz*. Además envía la transformada del modelo cinemático.

En la Figura 6 se puede observar cómo queda la estructura de los nodos con el visualizador *rviz* y el nodo aibo_server en funcionamiento.

[2]http://wiki.ros.org/rviz
[3]http://wiki.ros.org/urdf
[4]http://freecadweb.org/
[5]http://www.netfabb.com/

Figura 6: Estructuras de los nodos de ROS con aibo_server y el modelo visualizado con *rviz*.

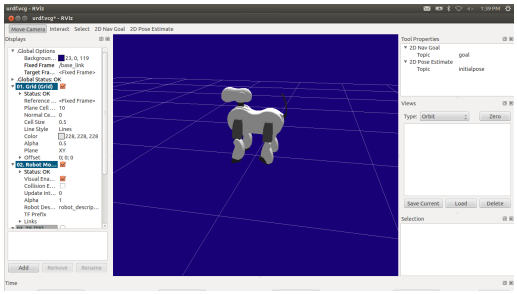

Figura 5: Modelo visto desde el framework de *rviz*.

5. Conclusiones y Trabajos Futuros

Como resultado del trabajo realizado, puede afirmarse que se ha alcanzado el objetivo de implementar los paquetes aibo_server y aibo_description que permiten la sincronización de varios robots AIBO. Estos paquetes están disponibles para su uso o modificación en https://github.com/Diamg/AiboRosPackages, con objeto de poder complementarlos y mejorar los resultados.

El módulo implementado permite adquirir los datos de los sensores y articulaciones de AIBO de forma satisfactoria a velocidades superiores a 10 Hz en forma de tópicos de ROS. Esta parte del módulo por si solo ya facilita enormemente la programación, el testeo y la depuración con el uso de las herramientas que facilita ROS. Además, en esta línea se ha incluido un modelo para el visualizador *rviz* con un aspecto visual y funcional complementando el módulo anterior.

Por otro lado, la parte implementada que se encarga de enviar las ordenes a AIBO permitiendo actuar sobre él, no ha dado unos resultados tan veloces como sería deseable y no permite el envío de trayectorias punto a punto con este método. De todos modos, URBI dispone de su propio generador de trayectorias por lo que permite enviar funciones de movimiento más complejas o posiciones concretas enviadas a un ritmo mas bajo del implementado.

Este proyecto deja un camino abierto a posibles mejoras y complementos. Entre otros, la integración del modulo de visión y audio usando URBI. Por otra parte, cabe la duda de si el comportamiento de envío sería mejor usando OPEN-R, lo que implica eliminar una capa de programación intermedia, URBI, que posiblemente está ralentizando y empeorando la comunicación entre cliente y servidor. En cuanto al modelo y el nodo que permite su integración con *rviz* podría completarse añadiendo otros sensores del AIBO como los acelerómetros, los infrarrojos, los sensores de presión y contacto, permitiendo, entre otros, realizar proyectos de odometría.

Referencias

[Arevalillo-Herráez *et al.*, 2011] Miguel Arevalillo-Herráez, Salvador Moreno-Picot, and Vicente Cavero Millán. Arquitectura para utilizar robots AIBO en la enseñanza de la inteligencia artificial. *IEEE-RITA*, 6(1):1–9, 2011.

[Baillie, 2005] Jean-C. Baillie. URBI: Towards a universal robotic low-level programming language. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 820–825, Aug 2005.

[Decuir *et al.*, 2004] John D. Decuir, Todd Kozuki, Victor Matsuda, and Jon Piazza. A friendly face in robotics: Sony's aibo entertainment robot as an educational tool. *Computers in Entertainment*, 2(2):14, 2004.

[Kecsap, 2010] Kecsap. Porting URBI 2.x for AIBO. biburlhttp://kecsapblog.blogspot.com.es/2010/12/porting-urbi-2x-for-aibo.html, 2010.

[Kertész, 2013] Csaba Kertész. Improvements in the native development environment for sony AIBO. *IJIMAI*, 2(3):50–54, 2013.

[Martínez Gómez, 2006] Jesús Martínez Gómez. Diseño de un teleoperador con detección de colisiones para robots AIBO. Master's thesis, Universidad de Castilla-La Mancha, Spain, 2006.

[Montero-Mora *et al.*, 2009] Ángel Montero-Mora, Gustavo Méndez-Muñoz, and José-Ramón Domínguez-Rodríguez. Metodologías de diseño de comportamientos para AIBO ERS7. Master's thesis, Universidad Complutense de Madrid, Spain, 2009.

[Morales-Jiménez, 2007] Jesús Morales-Jiménez. Localización de objetos y posicionamiento en el escenario deRoboCup Four-Legged con un robot AIBO. Master's thesis, Universidad de Castilla-La Mancha, Spain, 2007.

[Nardi and Lima, 2012] Daniele Nardi and Pedro Lima. RoboCup: the robot soccer world cup. In Pedro Lima and Rui Cortesao, editors, *Proceedings of the IROS-13 Workshop on Robot Competitions: Benchmarking, Technology Transfer and Education*. IEEE, 2012.

[Pérez *et al.*, 2010] Xavier Pérez, Cecilio Angulo, Sergio Escalera, and Diego E. Pardo. Vision-based navigation and reinforcement learning path finding for social robots. In *Jornadas ARCA 2010*, June 2010.

[Quigley *et al.*, 2009] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: An open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[Téllez *et al.*, 2006] Ricardo A. Téllez, Cecilio Angulo, and Diego E. Pardo. Evolving the walking behaviour of a 12

dof quadruped using a distributed neural architecture. In AukeJan Ijspeert, Toshimitsu Masuzawa, and Shinji Kusumoto, editors, *Biologically Inspired Approaches to Advanced Information Technology*, volume 3853 of *Lecture Notes in Computer Science*, pages 5–19. Springer Berlin Heidelberg, 2006.