Equilibrio del Robot AIBO usando DMPs*

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

Abstract

El trabajo presentado se enmarca en una iniciativa global que tiene como objeto la recuperación de la plataforma robótica AIBO de Sony. Para demostrar las prestaciones de la arquitectura propuesta y la viabilidad del robot AIBO como plataforma robótica útil, se han escogido algoritmos de aprendizaje por refuerzo muy novedosos como implementación en la tarea de mantener el equilibrio ante movimientos indeseados en la base de apoyo del robot. El robot AIBO puede ser controlado de forma permanente con un tiempo de respuesta adecuado para la tarea.

1. Introducción

Este trabajo forma parte de una iniciativa por fomentar la reutilización del robot AIBO de Sony. En esta dirección, la implementación de la tarea de mantener el equilibrio por parte de un robot cuadrúpedo, como es el caso del robot AIBO, sería factible a través de un gran número de algoritmos diferentes. Sin embargo, con objeto de demostrar la viabilidad del uso de robots AIBO en la robótica móvil actual, pese a la discontinuidad en su fabricación y servicio, se han escogido algoritmos, medios de comunicación y marcos de programación novedosos y del todo actuales.

Así, en los centros de investigación robótica de mayor nivel, como es el caso de *Boston Dynamics*, se está trabajando de un tiempo a esta parte con algoritmos DMPs (*Dynamic Movement Primitives*) [Schaal, 2006] como forma de creación de trayectorias en los robots móviles cuadrúpedos más avanzados [Raibert, 2008]. Se trata de algoritmos tales que, dada una demostración inicial de la dinámica aproximada del movimiento a realizar, adaptan la trayectoria original a la situación actual, generalmente cambiante, ya sea debido a una modificación en el punto inicial de movimiento, en el objetivo de movimiento o en obstáculos en el recorrido del movimiento. Una de las grandes ventajas en el uso de DMPs es la posibilidad de realizar el control de un robot sin tener información del modelo mecánico que lo representa.

*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.

Además, se ha decidido utilizar el entorno de trabajo de ROS (*Robotic Operative System*), que permite la creación de un enlace de comunicación entre el ordenador y el robot AIBO[1]. Se trata de un entorno de programación *open source* que se ha convertido en un estándar de facto en el mundo de la robótica móvil y se abre camino en los dominios de la robótica industrial.

Al robot AIBO se le ha dotado de un sensor triaxial de acelerometría, así como un giroscopio de tres ejes, con el objetivo de disponer de información del movimiento en el centro de gravedad del robot, el cual deseamos mantener estable. Con ayuda de las DMPs, junto con el algoritmo de aprendizaje por refuerzo PI^2 (*Path Integral Policy Improvement*) [Buchli *et al.*, 2011] se analizará la información recogida por los sensores inerciales, en especial el ángulo de inclinación del centro de gravedad del robot y se intentará su control para que se acerque lo más posible a la horizontal. Para la experimentación se ha utilizado una plataforma móvil de dos ejes controlada por un procesador Arduino, sobre la que se sitúa el robot AIBO. Al inclinarse la plataforma, el robot adapta la posición de sus motores para recuperar la posición de equilibrio.

A la hora de implementar, tanto las DMPs como el algoritme PI^2, se ha utilizado el código sobre ROS del *CLMC Lab*, en la *University of Southern California*[2]. Se trata de un código más complejo que el necesario para nuestra aplicación pues está pensado para el robot PR2 de Willow Garage [Bohren *et al.*, 2011]. Por ello, se ha adaptado al robot AIBO eliminando aquellos *packages* innecesarios y modificando otros. Además, se ha creado un ejecutable que permite el funcionamiento del robot con el conjunto de algoritmos implicados.

2. Estado del arte

En la investigación sobre estabilidad de robots, tanto bípedos como cuadrúpedos, existen mltitud de trabajos. A continuación se exponen un conjunto de antecedentes, organizados

[1] El *package* que permite esta coumincación ha sido diseñado inicialmente por el Dr. Ricado A. Téllez y posteriormente desarrollado en el Departamento de Automática de la Universitat Politècnica de Catalunya.

[2] La página principal es http://www-clmc.usc.edu/ y el github es https://github.com/usc-clmc/usc-clmc-ros-pkg

en diferentes ámbitos, que son de interés para el experimentación llevada a cabo.

2.1. Robot AIBO

El robot AIBO (***A**rtificial **I**ntelligence Ro**BO**t*) es un robot cuadrúpedo diseñado y fabricado por Sony Corporation, con apariencia canina. El *ERS-7* es el modelo elegido para el presente trabajo.

Este robot se considera autónomo en tanto que es capaz de extraer información de su entorno, funcionar por un largo período sin intervención humana, mover alguna de sus partes en un entorno de trabajo y, finalmente, evitar situaciones de peligro para las personas, los bienes y él mismo.

Los diseñadores del robot AIBO persiguieron el objetivo de conseguir que su comportamiento fuera lo más realista posbile. Para ello, avanzaron en diferentes caminos:

- Estímulos
  - Comportamientos reflexivos y deliberados según una escala de tiempo.
  - Comportamientos por órdenes externas y por deseos internos (instintos y emociones).
  - Motivaciones independientes generadas por partes del robot como cuello, cola y patas.
- Instintos y emociones con los que poder cambiar el comportamiento ante otros estímulos externos.
- Aprendizaje y evolución.

Las características hardware del robot pueden consultarse en [Anshar and Williams, 2007]. Las articulaciones, controladas mediante algoritmos *PID*, poseen rangos de trabajo diferentes según sea su función y las propias limitaciones físicas.

En cuanto al software, el robot AIBO funciona a partir de un sistema operativo denominado *Aperios*, desarrollado por Sony. Luego podría considerarse que posee varias capas de programario que permmiten el control del robot. Uno de los mayores problemas con el robot AIBO, y uno de los motivos del cese de su fabricación, es la dificultad para descentralizar los cálculos en un procesador externo.

2.2. Entorno de trabajo *ROS*

***R**obot **O**perating **S**ystem* [Robot Operating System, 2014] es un entorno de trabajo *open-source* y flexible para la programación de robots. Las bases de este proyecto se iniciaron con unas investigaciones en Stanford en 2007, cuando se llevaron a cabo diferentes prototipos de entornos de trabajo para programario de robots, como por ejemplo ***ST**anford **A**rtificial **I**ntelligence **R**obot* (*STAIR*) o ***P**ersonal **R**obotics* (*PR*). Más adelante, *Willow Garage*, una empresa inversora en robótica, proveyó de recursos para mejorar el concepto y permitir crear implementaciones correctamente testadas. Finalmente, con la colaboración desinteresada de numerosos investigadores, se ha mejorado el núcleo de *ROS* y las herramientas principales asociadas, convirtiéndose en una de las plataformas con más diversidad de algoritmos y más utilizada en las investigaciones de robótica.

Objetivos de *ROS*

El principal objetivo de *ROS* es poder *reutilizar* el código de desarrollo y de investigación en robótica. Ello se consigue mediante una estructura de procesos distribuidos. Otras características propias son además:

Descentralización. *ROS* está estructurado de forma que los processos están distribuidos, con la posibilidad de hallarse en *hosts* diferentes, pero funcionando conjuntamente.

Plurilingüismo. *ROS* se ha diseñado para ser neutral en cuanto al lenguaje de programación. Actualmente admite cuatro lenguajes de programación: *C++*, *Python*, *Octave* y *LISP*, habiendo ya otros en desarrollo.

Basado en herramientas. Se ha optado por diseñar un núcleo simple, sobre el que se utilizan multitud de herramientas para construir i hacer funcionar los diversos componentse de *ROS*.

Capa intermedia fina. En *ROS* se induce la independencia de los algoritmos con su núcleo creándolos en librerías separadas. Se facilita la extracción de código y su reuso.

Gratuito y *open-source*. El código nativo de *ROS* está disponible públicamente. Este hecho permite facilitar el testeo y corrección de *software* en todos los niveles.

Herramientas de *ROS*

ROS se basa, en gran parte, en la multitud de herramientas disponibles para tareas como, por ejemplo, navegar por el árbol de código fuente, obtener y establecer los parámetros de configuración, visualizar las conexiones entre procesos, medir el uso de ancho de banda, exponer de forma gráfica los datos de los mensajes. A continuación se listan algunas de las más utilizadas:

- `rviz`
  Entorno de visualización 3D que permite combinar los datos de los sensores del robot y el modelo.
- `rosbag`, `rxbag` y `rxplot`
  Permiten almacenar y reproducir los datos de un mensaje o un tópico, así como visualizarlos en forma gráfica.
- `rxgraph`
  Expone visualmente cómo funcionan los processos de *ROS* y sus conexiones.

2.3. *DMP* (*Dynamic Movement Primitive*)

Las *DMPs* representan un movimiento a partir de un conjunto de ecuaciones diferenciales, de forma que la dinámica del propio sistema sea capaz de corregir las perturbaciones que puedan aparecer. Además, la representación del movimiento permite modificar de forma sencilla el objetivo (o *goal*), así como la velocidad del movimiento. Esta robustez y adaptabilidad permiten mejorar los algoritmos utilizados en aprendizaje por demostración (*learning from demostration* o *LfD*).

El *LfD* podría estructurarse en tres grandes grupos, en función de cómo se adquieren los datos utilizados en la demostración:

Imitación. El ejemplo se realiza sobre una plataforma que no es el robot, por lo que la información extraída debe

ser modificada e interpretada para adecuarse a las articulaciones del robot.

Demostración. La ejecución se realiza sobre el mismo robot. Un método podría ser la tele operación del robot por parte del demostrador.

Trayectoria programada. La demostración se provee en forma de una serie de coordenadas de una trayectoria.

El sistema dinámico se puede interpretar como un sistema proporcional - derivativo (PD), con partámetros *K*, para el coeficiente proporcional, y *D*, para el derivativo. Las variables x y v corresponden a la posición y a la velocidad, la constante τ es el período de muestreo del movimiento y g es el parámetro de atracción del sistema.

$$\tau\dot{v} = K(g - x) - Dv + (g - x_0)f(s) \quad (1)$$

$$\tau\dot{x} = v \quad (2)$$

El sistema dinámico unidimensional (1)-(2) es estable, con punto final la posición g, para cualquier valor que tome $f(s)$, una función no lineal que no depende del tiempo, sino de una variable de fase $s \in [0, 1]$ que representa la duración del movimiento en tanto por uno.

$$f(s) = \frac{\sum_i w_i \psi_i(s) s}{\sum_i \psi_i(s)} \quad (3)$$

$$\tau\dot{s} = -\alpha s \quad (4)$$

La variable canónica posee una dinámica de desplazamiento definida por τ y por α, una constante pre-definida, como se aprecia en la ecuación diferencial (4). Además, la función $f(s)$ puede aprender con objeto de realizar movimientos complejos de forma arbitraria, de acuerdo al ajuste de los pesos w_i.

Las $\psi_i(s)$ son funciones base, en general gaussianas $\psi_i(s) = exp(-h_i(s - c_i)^2)$ que permitirán modificar la trayectoria inicial definida por los parámetros *K* y *D* (Figura 1), para así poderse adaptar a nuevas situaciones, como evadir obstáculos, cambio de punto final, etc.

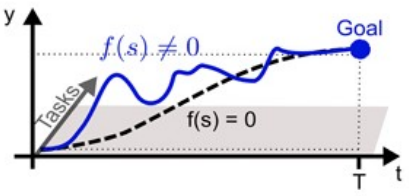

Figura 1: Representación de la *DMP* con función no lineal y sin ella.

Realmente, en las *DMPs*, lo que determina la trayectoria es la parte no lineal, y esta se ve controlada por los pesos w_i (ver Figura 2).

3. Estudios iniciales

3.1. Comunicación mediante *ROS*

El entorno de trabajo *ROS* es una herramienta que está al orden del día en robótica. Hasta el momento, no existía

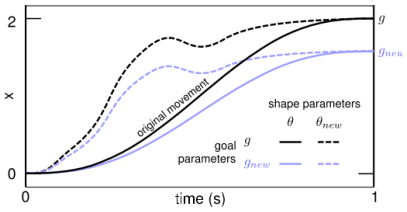

Figura 2: Efecto de los pesos w_i (representados por θ) y del objetivo (*goal*) en la trayectoria.

la posibilidad de comunicación del robot AIBO a través de *ROS*. Por ello, siguiendo una iniciativa del Departament d'Automàtica de la Universitat Politècnica de Catalunya, a través de diversos trabajos académicos, se ha promovido la realización de un `package` para permitir que el robot AIBO pueda trabajar con *ROS*. Gracias a ello, ahora es posible la transferencia de video, sonido y del control de las articulaciones del robot. Actualmente la conexión con AIBO comporta la creación de seis `topics` donde éste publica y dos en los que está como subscriptor:

- Publica en los `topics`:
  - **/aibo/infrared** Información de cada sensor de infra rojos (`distanceChest`, `distanceNear`, `distanceFar`), con la distancia calculada a un eventual obstáculo.
  - **/aibo/image** Publica las imágenes de la cámara frontal del AIBO, en formato `sensor_msgs::Image`.
  - **/aibo/joints** Información sobre las posiciones de cada una de las articulaciones en unidades de grados.
  - **/aibo/accel** Información del acelerómetre incorporado en el AIBO en unidades de m/s^2.
  - **/aibo/paws** Los sensores de contacto de las patas son binarios.
  - **/aibo/touch** Para los sensores de presión.
- Subscriptor de los `topics`:
  - **/aibo/sound** Donde se publican los sonidos a ser reproducidos per AIBO.
  - **/aibo/subJoints** Donde se informa a AIBO de la posición deseada para las articulaciones. El mensaje es de tipo `aibo_server::Joints`[3] con las unidades en grados.

Entre AIBO y *ROS*

El funcionamiento actual de la comunicación entre AIBO y *ROS* se basa en *URBI*. De igual forma, las acciones de control publicadas en *ROS* se interpretan y se transforman en comandos de *URBI*. Estas dos tareas se reproducen en paralelo a través de dos clientes de *URBI* por separado: (1) cliente publicador, que es el que introduce los datos de AIBO en los

[3]Este tipo de mensaje está incluído en el `package aibo_server`.

Figura 3: Comunicació entre l'AIBO, *URBI* i *ROS*

`topics`; (2) client subscriptor, con la función de recoger lo que se ha publicado en el `topic` de comandos.

Publicación en *ROS* de información de AIBO a través de *URBI*

1. *URBI* solicita la informació a AIBO, en forma de `callbacks`.
2. *URBI*, por otra parte, publica de forma continuada en los `topics`, donde hay subscriptores, el mensaje relacionado con el `topic`.
3. En el momento en que la información de AIBO es recibida por *URBI*, se modifica el mensaje que este publica en *ROS*.

Control de las articulaciones a partir de un `topic`

1. El algoritmo de control crea un `topic` denominado `aibo_server/aibo/subJoints`, y cada vez que se ha de modificar la posición de las articulaciones, publica un `message`[4].
2. Al aparecer un nuevo `message` en el `topic`, *URBI* toma el `message` y lo transforma para enviarlo como comando a las articulaciones de AIBO.

3.2. Entre el algoritmo de control y *ROS*

Con "algoritmo de control" nos referimos al código que procesa los cálculos que se requieren para llevar a cabo todo el proceso de estabilización. Este código se ejecuta en un ordenador o servidor, donde está instalado todo el *software* requerido.

El algoritmo de control sólo depende de *ROS* para recoger la información del conjunto acelerómetro-giroscopio y para ejecutar el movimiento de las articulaciones de AIBO. Por tanto, el `node` de ejecución es tanto subscriptor del `topic` del sensor, como, a la vez, es publicador del `topic` de los comandos para las articulaciones.

3.3. Entre el Arduino y *ROS*

El conjunt accelerómetro-giroscopio está conectado al Arduino, por lo que ha de ser éste quien cree el `topic` donde se publica la información que ha de transmitir por *ROS* al algoritmo de control. Para ello se utilizará el `package rosserial`.

4. Control de estabilidad

Intentar estudiar la estabilidad dinámica del robot [Yazdani *et al.*, 2012] implicaría un grado de complejidad demasiado elevado, por lo que el trabajo se ha limitado a la estabilidad estática [Hugel and Blazevic, 1999], es decir mantener el centro de gravedad *CdG* del robot dentro del polígono de estabilidad en todo momento.

Se han planteado dos tipos de aproximaciones al problema. La primera consiste en intentar mantener el *CdG* siempre en el mismo punto. La plataforma introduce una perturbación al sistema, desplazando el robot, quien ha de interpretar el desplazamiento e intentar corregirlo. Este planteamiento, como se verá, supera el grado de precisión conseguido por los sensores, por lo que será desestimada. La otra posibilidad, más sencilla, consiste en mantener la horizontalidad del cuerpo de AIBO. Es decir, el robot ha de ser capaz de reaccionar ante movimientos de la plataforma base con el objetivo de mantenerse en horizontal. Esta es la aproximación que se tomará.

Una forma de realizar el control de estabilidad podría ser basándose en un modelo de AIBO. Además de beneficiarse de la cinemática inversa, un modelo permiría aplicar algoritmos de control de estabilidad de forma virtual, en un simulador. Sin embargo, no existe en la literatura un modelo del sistema 'AIBO' lo bastante preciso como para poder trabajar con su estabilidad

Por ello, y con objeto de demostrar la viabilidad del entorno creado sobre un robot que ya no está en producción desde hace años, se ha determinado utilizar los algoritmos de *DMP*s, los cuales trabajan sin necesidad de conocer el modelo del sistema que están controlando y constituyen el estado del arte en aprendizaje robótico.

La creación del algoritmo de control de estabilidad basado en *DMPs* re realizará a partir del *package* de Scott Niekum, que está diseñado sobre el algoritmo modificado de [Pastor *et al.*, 2009]:

1. Dada una trayectoria, se calculan los pesos que la aproximan y se crea la *DMP*.
2. Proporcionadas las características del movimiento deseado (posición y velocidad actual, objetivo final, tiempo...), se realiza el cálculo de las acciones de control.
3. Para cada acción de control se comprueba si existe modificación del objetivo final o si se ha acabado de realizar el movimiento. En caso de cambiar de objetivo, se repite el paso anterior; en caso de haber finalizado, se para el robot y se procede a comprobar continuadamente si se genera un nuevo punto objetivo.

5. Implementación

La fotografía de conjunto del sistema implementado es la que puede observarse a la Figura 4.

Un requisito imprescindible para poder conseguir recuperar la estabilidad del robot es disponer de medidas de los movimientos del robot. Inicialmente se pensó en utilizar el sensor triaxial de accelerometría que lleva incorporado el propio AIBO, y de esta forma poder estudiar su movimiento. Sin embargo, unas primeras comprobaciones ya demostraron que la medida del ángulo de inclinación extraída no era fiable ni lo bastante precisa para la tarea a desarrollar.

5.1. Sensores

Por ello, se procedió a comprar un sensor externo triaxial de accelerometría con un giroscopio de tres ejes. El acelerómetro es un `MPU6050` y el giroscopio un `GY-521`. Se han escogido por estar diseñados para poder ser utilizados con Arduino. Además, existe mucha información sobre ellos,

[4] El `message` contiene un conjunto de `floats` que representan cada una de las articulaciones.

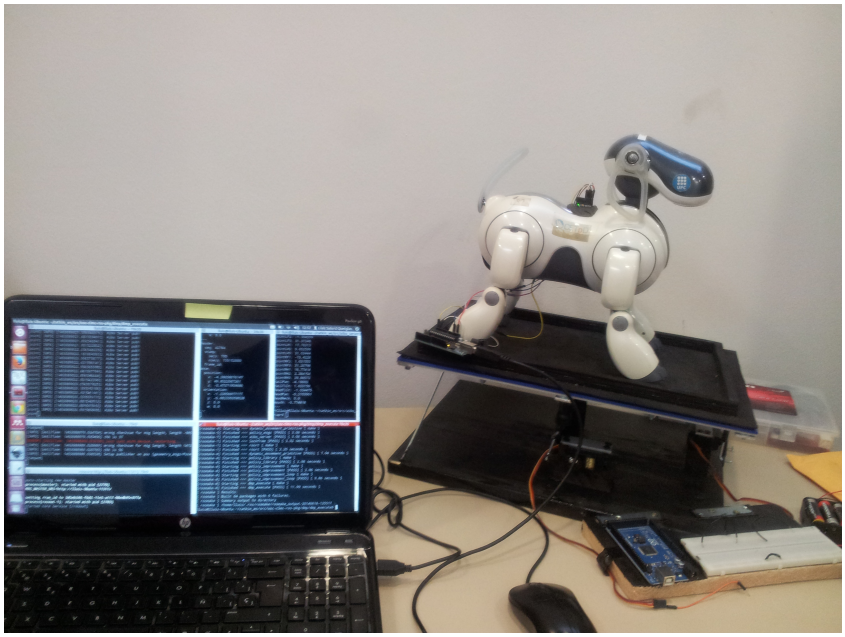

Figura 4: Fotografía del conjunto de elementos.

como librerías (`i2cdevlib`), aplicaciones realizadas, y es muy recomendable por su calidad-precio.

El cálculo de la inclinación se realiza utilizando el giroscopio como medida y filtrando los datos con ayuda de las medidas del accelerómetro, ya sea como un filtro de `Kalman` o como un filtro `complementario`. Ambos filtros no difieren demasiado en los resultados aportados, el Kalman siendo algo más preciso, pero requiere de mayores recursos computacionales.

Puesto que se cuenta con un *Arduino Uno* para realizar el procesado y la librería de *ROS* ya ocupa prácticamente la mitad de la memoria del microcontrolador, se ha optado por el filtro complementario,

$$\theta_{compl} = (1 - \lambda)(\theta_{compl} + w_{giro}\mathrm{d}t) + \lambda\theta_{accel} \qquad (5)$$

sobre el que se han realizado algunas mejoras. Para trabajar con la horizontalidad del robot ya es suficiente con la medida de la inclinación.

5.2. Comunicaciones

En el apartado 3.3 se explicó la forma de transmisión de la información desde Arduino a *ROS*. La comunicación entre el sensor y el microcontrolador se basará en el bus de comunicación $\mathrm{I}^2\mathrm{C}$, un bus bidireccional, pero que no permite enviar y recibir información de forma simultánea.

La comunicació con AIBO desde el ordenador donde corre el algoritmo de control de estabilidad se realiza a través de un *Access Point* (*AP*). El tipo de cifrado de la contraseña debe ser `WEP` o sin contraseña.

Para configurar la tarjeta de red de AIBO se ha de acceder al fichero de configuración de la tarjeta de memoria `OPEN-R\SYSTEM\CONF\WLANDFLT.TXT`. Una posible configuración es la que se expone a continuación, [Téllez, 2004]

```
HOSTNAME=AIBO
ETHER_IP=192.168.10.124
ETHER_NETMASK=255.255.255.0
IP_GATEWAY=192.168.10.1
ESSID=AIBONET
WEPENABLE=1
WEPKEY=AIBO2
APMODE=2 # this mode indicates auto-mode
CHANNEL=3
```

5.3. Procesos iniciales

Para el buen funcionamiento de las *DMPs* es necesario iniciar una serie de procesos que permitan el intercambio de información y así poner en funcionamiento el conjunto de sistemas.

Primero se inicia la carga del programa de control del *CdG* en Arduino. En el GitHub `github.com/lluissalord/TFG` puede recuperarse un programa de Arduino (`arduino/MPU6050_compl.ino`) que debe ser instalado en la placa para poder transmitir

de forma correcta la información en el `topic` de *ROS* correspondiente (`/pos`).

Parar poder realizar la carga del programa, se ha de tener conectado el Arduino con un cable `USB` al ordenador, se ha de abrir con el `IDE` de Arduino el fichero mencionado y realizar la opción `Archivo->Subir`.

Una vez dispuesto el Arduino, viene la fase de ejecución por línea de comandos, las cuales podrían integrarse en un fichero *launch*. Cada una de las acciones seguientes debe realizarse por separado en un terminal independiente, preferiblemente en este orden:

$ roscore ejecuta el *ROS* `Master`.

$ rosrun rosserial_python serial_node.py \dev\ttyACM0 activa la comunicación entre el Arduino y *ROS*. Podría darse el caso que la entrada de datos no sea `\dev\ttyACM0`; para corroborarlo se ha de observar en el `IDE` de Arduino a qué puerto está conectado.

$ rosrun aibo_server aibo_server 192.168.0.124 ejecución del programa que permite la comunicación AIBO-*ROS*. El número 192.168.0.124 es la `IP` del robot AIBO, la cual varía según la configuración que se haya seguido en la fase de set-up de la conexión inalámbrica.

$ rosrun dmp_execute dmp_execute 0 inicia el algoritmo de control. El número final varía entre 1 y 0, según si se desea utilizar el algoritmo de aprendizaje PI^2 o no, respectivamente.

5.4. Funcionamiento de *DMPs* con PI^2

El funcionamiento con el uso del algoritmo PI^2 varía significativamente respecto a las *DMPs* originales. El algoritmo PI^2 ha de permitir un proceso de aprenendizaje iterativo hacia el objetivo fijado, sin cambiar las condiciones del entorno. Aunque el algoritmo ha sido generado, los resultados no fueron satisfactorios, por lo que se descartó su uso en este trabajo.

6. Resultados, conclusiones y trabajos futuros

Puede considerarse que los resultados conseguidos son muy satisfactorios ya que se han cubierto los objetivos previstos.

A nivel global, se ha podido realizar la optimización de la adaptabilidad del robot a un entorno accesible y desconocido.

A nivel de objetivos específicos, en cuanto a accesibilidad del trabajo para futuras extensiones, en tanto que englobado en la iniciativa de recuperación del robot AIBO de Sony, se ha actualizado continuadamente una plataforma de desarrollo colectivo de *software* GitHub. Toda la faena realizada puede consultarse y descargarse desde `https://github.com/lluissalord/TFG/`. En referencia a la implementación del entorno de comunicación, se ha mejorado el `package` inicial. Se ha podido también demostrar cómo esta plataforma permite la implementación de algoritmos de control que son estado del arte a nivel de investigación. Por otra parte, se ha realizado un estudio de los moviments del robot a partir de un sensor externo triaxial de accelerometría y un giroscopio de tres ejes. Las inclinaciones se han registrado perfectamente, con una precisión superior a la requerida para los cálculos. Sin embargo, la información de los desplazamientos calculada a partir del accelerómetro no es fiable.

Se pueden, además, plantear varias líneas de trabajo: (1) mejoras en la comunicación AIBO-*ROS*, la cual provoca actualmente bloqueos temporales; (2) implementación del algoritmo PI^2 de forma más integrada; (3) plataforma automatizada para facilitar el aprenendizaje del algoritme anterior manteniendo una inclinación constante; (4) conseguir extraer el desplazamiento del *CdG* gracies al uso de algoritmos de visión; (5) mejorar el modelo del robot AIBO.

Referencias

[Anshar and Williams, 2007] Muh. Anshar and Mary-Anne Williams. Extended Evolutionary Fast Learn-to-Walk Approach for Four-Legged Robots. *Journal of Bionic Engineering*, 4(4):255–263, December 2007.

[Bohren *et al.*, 2011] Jonathan Bohren, Radu Bogdan Rusu, Edward Gil Jones, Eitan Marder-Eppstein, Caroline Pantofaru, Melonee Wise, Lorenz Mösenlechner, Wim Meeussen, and Stefan Holzer. Towards autonomous robotic butlers: Lessons learned with the PR2. In *ICRA*, pages 5568–5575. IEEE, 2011.

[Buchli *et al.*, 2011] Jonas Buchli, Freek Stulp, Evangelos Theodorou, and Stefan Schaal. Learning variable impedance control. *International Journal of Robotics Research*, 2011.

[Hugel and Blazevic, 1999] Vincent Hugel and Pierre Blazevic. Towards efficient implementation of quadruped gaits with duty factor of 0.75. *Robotics and Automation, 1999. ...*, (May), 1999.

[Pastor *et al.*, 2009] Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. *2009 IEEE International Conference on Robotics and Automation*, pages 763–768, May 2009.

[Raibert, 2008] Marc Raibert. BigDog, the Rough-Terrain Quadruped Robot. In Myung J. Chung, editor, *Proceedings of the 17th IFAC World Congress*, volume 17, 2008.

[Robot Operating System, 2014] Robot Operating System. ROS.org, 2014. Fecha de consulta: 22/02/2014.

[Schaal, 2006] Stefan Schaal. Dynamic Movement Primitives -A Framework for Motor Control in Humans and Humanoid Robotics. In Hiroshi Kimura, Kazuo Tsuchiya, Akio Ishiguro, and Hartmut Witte, editors, *Adaptive Motion of Animals and Machines*, pages 261–280. Springer Tokyo, 2006.

[Téllez, 2004] Ricardo A. Téllez. Aibo Quickstart Manual. Technical report, Grup de Recerca en Enginyeria del Coneixement (GREC), 2004.

[Yazdani *et al.*, 2012] Reza Yazdani, Vahid Johari Majd, and Reza Oftadeh. Dynamically stable trajectory planning for a quadruped robot. *20th Iranian Conference on Electrical Engineering (ICEE2012)*, pages 845–850, May 2012.