# Conectando el Robot AIBO a ROS: Extracción de imágenes*

**Lucía Lillo-Fantova, Ricardo A. Téllez, Manel Velasco, Cecilio Angulo**
Universitat Politècnica de Catalunya · UPC BarcelonaTech
Pau Gargallo 5. ESAII – Departament d'Automàtica, Barcelona, Spain 08028
lucialillofantova@gmail.com, tellezatwork@gmail.com,
{manel.velasco, cecilio.angulo}@upc.edu

## Abstract

En este documento se realiza una introducción al trabajo realizado con el robot AIBO, de Sony, para la obtención de imágenes, así como su tratamiento con el objetivo que el robot navegue de forma autónoma mediante algoritmos de visión artificial. A nivel práctico, el principal punto de interés del artículo se centra en el entorno de programación, que se ha exportado al marco de ROS (*Robot Operating System*), de forma que se pueda hacer uso de la extensa biblioteca de algoritmos ya desarrollados en este entorno.

## 1. Introducción

Este proyecto surge a partir de la voluntad de recuperar la funcionalidad del robot AIBO de Sony [Decuir *et al.*, 2004], actualmente discontinuado en su producción y soporte, dentro de un entorno de programación actualizado y potente [Kertész, 2013], así como dotarlo de las herramientas necesarias [Kolovrat, 2013] para proporcionarle la capacidad de navegar autónomamente en un entorno de trabajo mediante algoritmos de visión artificial.

Como se muestra en la Figura 1, a partir del sistema operativo de AIBO, OPEN-R SDK, se ha integrado el entorno de programación URBI [Baillie, 2005a; 2005b], mediante el cual se enviarán datos a través de una estructura servidor/cliente entre el robot, *servidor*, y un ordenador externo, *cliente*. Los datos recibidos por el cliente deberán ser almacenados en un buffer temporal para su posterior tratamiento mediante algoritmos en OpenCV [Bradski, 2000], encargado de su transformación en imágenes interpretables por ROS [Quigley *et al.*, 2009]. La voluntad de integrar ROS en el proyecto nace como una oportunidad de poder hacer uso de la gran cantidad de librerías y herramientas que incorpora este sistema operativo para facilitar la creación de aplicaciones robóticas. Entre ellas, nos ofrece herramientas como SLAM [Huang and Dissanayake, 2007; Riisgaard and Blas, 2005], SIFT [Flores and Braun, 2011] o RANSAC, de gran utilidad para el proyecto.

*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.

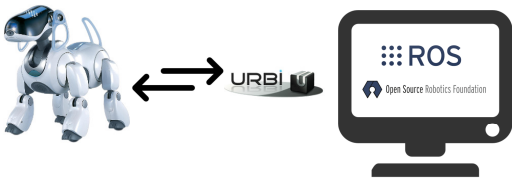


Figura 1: Estructura del modelo de integración del robot AIBO de Sony en ROS.

A partir de aquí se definen una serie de objetivos concisos a seguir durante el desarrollo del trabajo.

- Comparación de las frecuencias de ciclo de ROS para mejorar tanto la frecuencia de publicación de las imágenes como del resto de sensores y actuadores.
- Optimización del tamaño de los buffers para la mejora de la frecuencia de publicación.
- Reducir el retraso en el procesado de las imágenes para mejorar la toma de decisiones en tiempo real.

A partir de esta conexión y mediante algoritmos de visión artificial se habrá desarrollado una base óptima de programación para el funcionamiento del robot AIBO en tiempo real.

## 2. El robot AIBO

AIBO es un robot cuadrúpedo creado por SONY e introducido en 1999 con el modelo ERS-110 [Téllez, 2004]. Fue el primer robot de este tipo en el mercado. Inicialmente SONY enfocó este producto como un robot de entretenimiento para el uso doméstico, pero tuvo una gran repercusión entre la comunidad científica puesto que facilitaba la investigación en campos como la inteligencia artificial y la interacción entre robots [Zhang and Chen, 2007].

SONY comenzó su comercialización limitando el acceso al lenguaje de programación a ellos mismos y a los participantes de la RoboCup. En 2001 retiró los derechos de autor al kit de programación del AIBO permitiendo así el uso no comercial de éste. El kit incluía los lenguajes de programción R-CODE, OPEN-R SDK y el Framework remoto de AIBO. Además de las propias plataformas oficiales de SONY

la comunidad de usuarios desarrolló toda una serie de plataformas sobre el robot para facilitar la programación de éste, tales cómo Tekkotsu [Tek, 2014], URBI [Gostai, 2014] o la compilación cruzada de Webots [Hohl *et al.*, 2006].

El proyecto ideado por SONY únicamente duró siete años, parando la producción en 2006, pero a pesar de la brevedad lanzó toda una serie de modelos en tres generaciones de robot (Figura 2). Pese a la interrupción de la fabricación hoy en día aún se encuentran eventos tales cómo la Convención Internacional de AIBO donde se realizan intercambios de software open source.

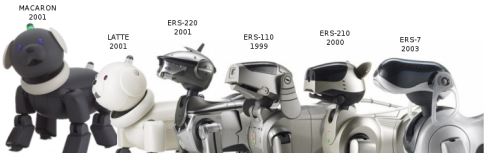


Figura 2: Modelos de AIBO en el período 1999-2006.

## 2.1. Hardware

AIBO presenta un microprocesador de 64 bits funcionando a una frecuencia de 576MHz. La memoria del robot está dividida en dos elementos, una memoria RAM interna para el desarrollo de funciones de 64MB y una memoria externa de 32MB donde se encuentran los programas a ejecutar por el robot y que el usuario puede modificar.

Como sensores, dispone de:

- un sensor de presión en cada una de las patas (*pawLF, pawLH, pawRF, pawRH*) y otro en el mentón (*chinSensor*), un sensor electrostático en la cabeza (*headSensor*) y otros tres en el lomo (*backSensorF, backSensorM, backSensorR*), así como un sensor de temperatura y un sensor de vibración.
- una cámara CMOS (*camera*) con parámetros regulables como la resolución y el formato de la imagen; infrarrojos para la detección de la distancia, uno en el pecho (*distanceChest*), además de otros dos en la cabeza, (*distanceNear*) y (*distance*).
- un micrófono estéreo en cada oreja con un buffer de 2048 bytes donde se encuentra el último sonido captado.
- tres acelerómetros (*accelX, accelY, accelZ*).

El robot controla toda una serie de actuadores que afectan al entorno:

- cuatro patas, cada una con tres articulaciones, por tanto doce motores en total para el control de las extremidades; tres grados de libertad sobre la cabeza; inclinación de las orejas; dos grados de libertad en la cola; control de la obertura de la boca. Resultando todo en un total de 20 motores.
- red Wi-Fi; micrófono para reproducir sonidos guardados previamente en su Memory Stick; un conjunto de LEDS para comunicar el estado del robot.

## 2.2. Software

El software OPEN-R SDK es el que permite mayor libertad de programación de entre los subministrados por Sony.

OPEN-R es un lenguaje open source basado en C++ de bajo nivel que permite el acceso y modificación de cualquier variable del robot. Está fundamentado en la creación de objetos que se comunican entre ellos mediante mensajes entre sujetos y observadores (Figura 3).

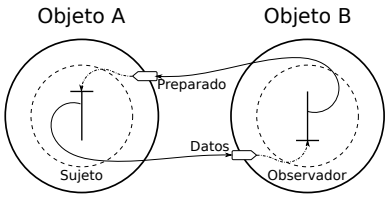


Figura 3: Comunicación entre objetos con OPEN-R.

Con posterioridad se desarrolló Tekkotsu, una aplicación open source para el diseño de programas en AIBO. Maneja tareas de bajo nivel y está completamente desarrollada sobre C++. Con el objetivo que el movimiento del robot AIBO sea lo más fluido posible, se diferencia entre el proceso principal *Main*, donde se realizan los procesos relacionados con la aplicación, y el proceso de captación de datos del AIBO *MotionManager*. La estructura que comunica estos dos procesos está dada por el *MotionCommand* y se encuentra en una zona de la memoria compartida por ambos.

URBI, por otra parte, es un software open source generalista para el control de robots y sistemas complejos. Está diseñado para realizar tareas en paralelo y está dirigido por eventos, dos de los principales lemas en la programación de sistemas complejos. Otro de los principales objetivos de este lenguaje es la compatibilidad entre robots facilitando así la tarea de programación, pudiendo utilizar los mismos objetos para diversos robots.

URBI presenta una librería de componentes **UObject** escritos en C++ que incorpora un interfaz de programación de robots que describe motores, sensores y algoritmos estándars de robots. Estos objetos pueden ser usados dentro del **urbiscript**, como elementos locales, o bien como elementos remotos directamente sobre Windows, Linux o Mac OSX para tener una mayor capacidad de cálculo.

**Urbiscript** es el encargado de orquestrar los diversos componentes para que interactuen entre ellos generando así funciones de alto nivel, pero con los dos lemas principales de la robótica nombrados anteriormente en mente.

La elevada compatibilidad de este software se debe a la estructura cliente/servidor de URBI que permite la implementación dentro de cualquier lenguaje de programación que tenga la capacidad de crear y trabajar con sockets TCP.

La última versión compatible con el robot AIBO es URBI 1.5, que no permite la implementación directa de URBI como objeto de ROS (sí con URBI 2.1). Es por esto que se trabajará con un cliente de URBI que será quien envíara los datos a ROS.

## 3. Integrando ROS en URBI

ROS, *Robot Operating System*, incorpora gran cantidad de librerías y herramientas para facilitar la creación de aplicaciones robóticas. En consecuencia se ha optado por la integración de las dos plataformas. Para ello se ha generado un script de C++ donde se define un cliente de URBI, el ordenador, que será el encargado de recibir y enviar los datos al servidor, el robot. Para que el robot AIBO sea capaz de interactuar con URBI se ha incorporado un objeto de URBI en la memoria programable de éste (Memory stick).

La transmisión de datos desde el robot AIBO al cliente de URBI se realiza en primer lugar mediante el establecimiento de una conexión cliente-servidor con un socket TCP/IP. Posteriormente se realizarán callbacks a las funciones especificadas por el cliente y que devolverán los datos solicitados. Finalmente, el cliente también podrá enviar órdenes a AIBO.

Paralelamente, también se ha creado un nodo de ROS dentro del cuál se dispone de los tópicos correspondientes a los diferentes actuadores y sensores. Para finalizar la tarea de publicación de los datos se debe crear una serie de publishers que publiquen la información en el tópico correspondiente, es decir, envíen los datos recibidos por el cliente al nodo de ROS creado. Por último, sólo queda cubrir el envío de órdenes a través de ROS el cuál será realizado con la creación de subscribers al tópico que se desee, quienes a su vez enviarán la información al cliente. Éste también enviará la información al servidor, AIBO, el cuál finalmente las llevará a cabo.

## 4. Obtención de imágenes

El robot AIBO dispone de una cámara de tecnología CMOS que presenta una resolución de 350000 píxels (480 x 720) [Pérez *et al.*, 2010a; Pérez *et al.*, 2010b]. Las imágenes captadas tendrán que ser convertidas para permitir el procesamiento aplicado posteriormente.

### 4.1. Estructura de las Imágenes

El primer punto a tratar es la transformación de las imágenes a partir de URBI en un formato adecuado para la publicación en ROS. El formato correspondiente a los tópicos de imagen en ROS es *sensor_msgs::Image*. Para realizar la conversión se dispone de la plataforma *cv_bridge*, una librería de OpenCV que facilita la conversión entre imágenes de OpenCV a mensajes de ROS. Para esto se debe transformar igualmente la imagen a *cv::Mat o IplImage* según la aplicación pero representan el mismo formato. En consecuencia el primer paso será la transformación de los datos a OpenCV.

Primero se operan los mensajes de URBI de tipo *UMessage* donde si el contenido es correcto contendrá un apuntador al valor del mensaje. Los datos adquiridos serán de tipo *UValue* el cuál contiene varias variables.

- **UDataType** Contiene la información sobre el tipo de datos enviados.
- **UDictionary*** Contiene los datos si son de tipo DATA_DICTIONARY.
- **urbi::UBinary** Contiene los datos enviados si éstos son de tipo DATA_BINARY. Concretamente representa el caso de la transmisión de imágenes (Cuadro 1).
- **ufloat** Contiene los valores si los datos enviados son del tipo DATA_DOUBLE. Es el caso de todos los datos enviados por el AIBO a excepción de los mensajes de sonido e imagen, es decir, los valores de los sensores y actuadores.
- **void*** Variable interna que apunta a la zona de almacenaje.
- **std::string** Si la información enviada es de tipo DATA_STRING se le asocia este tipo.

De entre ellas se seleccionan los datos del tipo DATA_BINARY, representados por mensajes de tipo *UBinary* (Figura 4). Finalmente, se escoge la información del subtipo *UImage*.

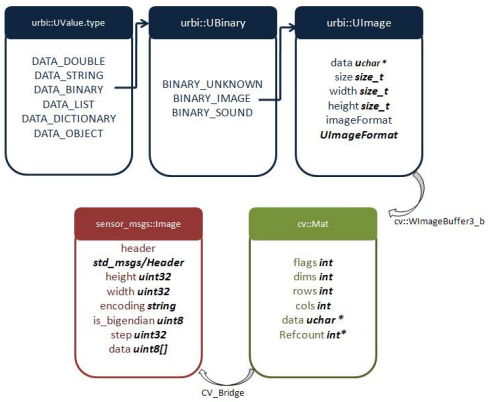


Figura 4: Obtención de las imágenes.

| | | |
|---|---|---|
| BINARY_UNKNOWN | struct{<br>void* data<br>size_t size<br>}common | |
| BINARY_IMAGE | data | unsigned char* |
| | size | size_t |
| | width | size_t |
| | height | size_t |
| | imageFormat | UImageFormat |
| BINARY_SOUND | data | char |
| | rate | size_t |
| | size | size_t |
| | sampleSize | size_t |
| | channels | size_t |
| | sampleFormat | USoundSampleFormat |
| | soundFormat | USoundFormat |

Cuadro 1: Tipos de información dentro de la variable *UBinary*.

Esta información será guardada en un buffer de 4 imágenes ya en formato jpg, por lo que se presentará una imagen

de 3 canales *R, G, B* que posteriormente será abierta mediante OpenCV y transformada en una imagen de tipo *cv::Mat*, tipo obligado para la implementación de las funciones de la librería de *CV_Bridge*. Mediante ésta transformaremos las imágenes a un formato interpretable por ROS y que serán enviadas al tópico de imagen correspondiente.

## 5. Resultados

En este apartado se realizará un estudio sobre la velocidad de envío de imágenes entre el robot AIBO y el ordenador donde corre en núcleo de ROS. La frecuencia de publicación de imágenes viene limitada principalmente por la capacidad del propio robot para el envío de imágenes. Se ha de considerar que éste se dejó de fabricar en 2006 y que el último modelo fue diseñado en 2005, por tanto existen limitaciones de hardware que son insalvables en la comunicación cliente-servidor como son el microprocesador, el módulo wi-fi y la calidad de la imagen.

Inicialmente se ha realizado un primer estudio con un rango de frecuencias de funcionamiento del nodo de ROS cercanas al valor deseado para la publicación de imágenes que estaría entorno a los 25Hz.

Tal y como se muestra en la Figura 5, no se ha obtenido el comportamiento esperado debido a las propias limitaciones del entorno, provocando así caídas en el envío de datos. Conjuntamente a esto, se detecta una tendencia negativa de la frecuencia media de publicación. Esta tendencia se justifica ya que la conexión cliente/servidor es posterior al encendido del AIBO, hecho que provoca que el buffer interno del robot se encuentre lleno inicialmente. En consecuencia, en el instante inicial, la frecuencia es la deseada pero posteriormente no es capaz de adquirir las imágenes a este ritmo ya que el robot no es capaz de funcionar a esta frecuencia.

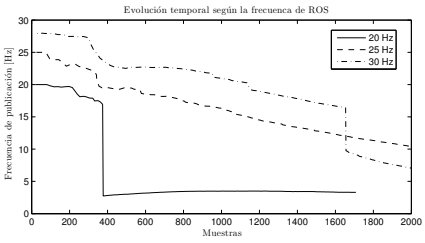


Figura 5: Comparativa de las frecuencias de publicación entorno los 25Hz.

Por todo ello, se realiza un estudio de frecuencias acorde al ritmo de trabajo del robot, entorno a los 7.5 Hz, reflejado en la Figura 6.

Finalmente se ha procedido a disminuir la calidad de la imagen comprimida (jpeg) que envía el robot AIBO para mejorar la velocidad de transmisión. Se ha definido una compresión del 80 %.

Tal y cómo se puede observar en la Figura 7 se detecta una mejoría, que sería más destacable con un factor de compresión mayor pero esto implicaría una calidad de imagen muy

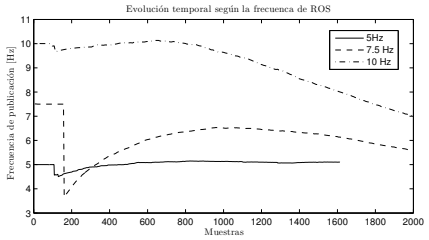


Figura 6: Comparativa de las frecuencias de publicación entorno los 7.5Hz.

baja para ciertas aplicaciones. Debido a esto no se ha procedido a estudios posteriores con medidas más elevadas de comprensión de la imagen.

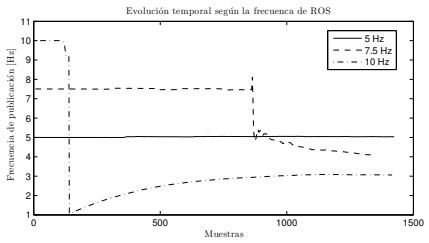


Figura 7: Comparativa de las frecuencias de publicación entorno los 7.5Hz, con la imagen jpg comprimida al 80 %.

## 6. Conclusiones

En este documento se documenta el trabajo realizado con el robot AIBO, de Sony, para la obtención de imágenes, así como su tratamiento con el objetivo que el robot navegue de forma autónoma mediante algoritmos de visión artificial. A nivel práctico, el principal punto de interés del artículo se centra en el entorno de programación, que se ha exportado al marco de ROS (*Robot Operating System*), de forma que se pueda hacer uso de la extensa biblioteca de algoritmos ya desarrollados en este entorno.

En el apartado de experimentación se ha observado que la frecuencia óptima de trabajo del nodo de ROS se debería encontrar entre los 5 y los 7.5Hz para una frecuencia de publicación constante y un nivel de compresión de la imagen del 80 % para mantener unos mínimos estándares de calidad. Este valor se encuentra alejado de los valores óptimos de trabajo, que se encuentran entorno los 25Hz, lo que comportará limitaciones sobre los algorismos de tratamiento de imágenes a realizar sobre AIBO. Futuros trabajos debería incidir en evitar que la transmisión de datos sea el principal cuello de botella del proceso.

## Referencias


[Baillie, 2005a] Jean-Christophe Baillie. *URBI Language Specification*. 2005.

[Baillie, 2005b] Jean Christophe Baillie. URBI: Towards a universal robotic low-level programming language. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 3219–3224, 2005.

[Bradski, 2000] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[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.

[Flores and Braun, 2011] Pablo Flores and Juan Braun. Algoritmo SIFT : fundamento teórico. pages 1–5, 2011.

[Gostai, 2014] Gostai. URBI Doc for Aibo ERS2xx ERS7 and URBI 1.0, 2014. Data de consulta: 6 de març de 2014.

[Hohl *et al.*, 2006] Lukas Hohl, Ricardo Tellez, Olivier Michel, and Auke Jan Ijspeert. Aibo and Webots: Simulation, wireless remote control and controller transfer. *Robotics and Autonomous Systems*, 54(6):472–485, June 2006.

[Huang and Dissanayake, 2007] Shoudong Huang and Gamini Dissanayake. Convergence and Consistency Analysis for Extended Kalman Filter Based SLAM. *IEEE Transactions on Robotics*, 23(5):1036–1049, October 2007.

[Kertész, 2013] Csaba Kertész. Improvements in the native development environment for Sony AIBO. *International Journal of Interactive Multimedia and Artificial Intelligence*, 2(3):51, 2013.

[Kolovrat, 2013] Stipe Kolovrat. Development of Android Application for Sony Aibo ERS-7 robot. Master's thesis, Universitat Politècnica de Catalunya, 2013.

[Pérez *et al.*, 2010a] Xavier Pérez, Cecilio Angulo, and Sergio Escalera. *Vision-based Navigation and Reinforcement Learning Path Finding for Social Robots*. PhD thesis, Universitat Politécnica de Catalunya, 2010.

[Pérez *et al.*, 2010b] 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.

[Riisgaard and Blas, 2005] Søren Riisgaard and Morten Rufus Blas. Slam for dummies: A tutorial approach to simultaneous localization and mapping. Technical report, 2005.

[Tek, 2014] Tekkotsu, 2014. Fecha de consulta: 18/06/2014.

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

[Zhang and Chen, 2007] Jiaqi Zhang and Qijun Chen. Learning based gaits evolution for an AIBO dog. *2007 IEEE Congress on Evolutionary Computation*, pages 1523–1526, September 2007.