Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXXIII, págs. 2595-2618 (artículo completo)
Graciela Bertolino, Mariano Cantero, Mario Storti y Federico Teruel (Eds.)
San Carlos de Bariloche, 23-26 Setiembre 2014

EVALUACIÓN DE UNA PLATAFORMA BASADA EN MENSAJES PARA LA OPERACIÓN REMOTA DE PROCESOS

Sebastián J. Tosco[a], **Fernando Corteggiano**[b]

[a]*Grupo de Investigación y Desarrollo Aplicado a las Telecomunicaciones (GIDAT), Universidad Nacional de Río Cuarto, Ruta Nacional 36 km 601, 5800 Río Cuarto, Cba., Argentina, stosco@ing.unrc.edu.ar, http://www.ing.unrc.edu.ar/gidat*

[b]*Grupo de Investigación y Desarrollo Aplicado a las Telecomunicaciones (GIDAT), Universidad Nacional de Río Cuarto, Ruta Nacional 36 km 601, 5800 Río Cuarto, Cba., Argentina, fcorteggiano@ing.unrc.edu.ar, http://www.ing.unrc.edu.ar/gidat*

Keywords: ROS, middleware, laboratorio remoto, teleoperación.

Resumen. Hoy en día la redes de telecomunicaciones son transversales a numerosas disciplinas relacionadas con la ciencia y la ingeniería. Esto conlleva a que el modelo de pensamiento y diseño haya cambiado en las últimas décadas y que hablar de teleoperación, telemedicina y control a distancia de aplicaciones agroindustriales sea común en los ámbitos académico-industriales.

Cuando se plantea este tipo de sistema, se hace necesario un middleware (capa intermedia de software) que facilite la abstracción de los detalles de la red y sus parámetros para centrarse en lo que relevante desde el punto de vista del diseño y operación. Por tanto, desde el punto de vista del operario, la operación remota de un proceso/sistema no debería ser sustancialmente diferente en comparación con el manejo local del mismo.

Existen varios tipos de plataformas para este fin, una de estas son los Middleware Orientados a Mensajes (MOM). Éstos proporcionan comunicación distribuida sobre la base del modelo de interacción asíncrona; este modelo sin bloqueo permite resolver muchas de las limitaciones que se encuentran en otras soluciones. Los participantes en un sistema basado en MOM no están obligados a bloquear y esperar a enviar un mensaje, se les permite continuar con el proceso una vez que un mensaje ha sido enviado. Esto permite la entrega de mensajes cuando el remitente o el receptor no está activo o disponible para responder en el momento de la ejecución. En los sistemas distribuidos basados en MOM se ofrece una comunicación entre procesos orientada a la prestación de servicios.

En el presente trabajo se han realizado pruebas sobre un MOM particular llamado Robot Operating System (ROS) el cual es un meta-sistema operativo de código abierto pensado para robótica que corre con todo su potencial sobre plataformas basadas en UNIX. ROS ofrece los servicios que se esperarían de un sistema operativo, incluida la abstracción de hardware de bajo nivel de control del dispositivo, la aplicación de la funcionalidad de uso común, el paso de mensajes entre los procesos y la gestión de paquetes. También proporciona herramientas y bibliotecas para la obtención, construcción, escritura y ejecución de código en varios equipos.

El objetivo que se persigue en el presente trabajo es analizar el comportamiento de ROS como capa intermedia para operar remotamente un sistema y evaluar su desempeño ante diversas situaciones de congestión de la red de telecomunicaciones.

1 INTRODUCCIÓN

Con el avance de la tecnología en el campo de la computación y las redes, el estudio de la interacción entre las comunicaciones y los sistemas de control se ha vuelto atractivo para la investigación y el desarrollo. Pero esto no es todo. En el ámbito educativo, también existe mucho interés en dicha integración, especialmente en las áreas afines a la enseñanza de las ciencias y la ingeniería. Siendo así, los laboratorios experimentales representan una herramienta de enseñanza fundamental.

Sin embargo, dado que el concepto tradicional de laboratorio de prácticas, en el marco de las enseñanzas de tipo científico o tecnológico, plantea en la actualidad una serie de dificultades (tanto económicas como de infraestructura), hoy en día se han empezado a desarrollar laboratorios teleoperados a través de redes de telecomunicaciones.

Se puede definir un laboratorio teleoperado o remoto como el conjunto de equipos e instrumentos reales que se pueden operar y controlar remotamente, utilizando una interfaz específica. La idea de estos dispositivos no es nueva, dado que se viene utilizando desde hace décadas en la industria. Sin embargo, el advenimiento de nuevas tecnologías en lo que a redes y computadores se refiere genera una constante actualización en las estructuras y configuración de los equipos.

Por otro lado, la aplicación de técnicas matemáticas al análisis de datos e imágenes ha permitido que este campo de la investigación tenga alta aplicabilidad, tanto en el ambiente académico como en la industria. De hecho, estudios recientes sobre la posibilidad de medir temperaturas mediante una cámara de video incluso en circunstancias donde la utilización de un sensor es inviable, abre las puertas del pensamiento al abanico de posibilidades que presenta el procesamiento de imágenes. También, el análisis de datos provenientes de múltiples sensores y la toma de decisiones para realizar un control óptimo sobre un sistema son hoy en día temáticas muy bien estudiadas.

Por último, la evolución de las redes de telecomunicaciones permite la implementación de transmisión de datos en tiempo real. En la actualidad es posible disponer de ancho de banda razonables a precios módicos, lo que permite el acceso remoto a equipos e instalaciones de un laboratorio real.

En el marco descrito entonces, el problema se puede formular así: ¿Es posible proponer una arquitectura para LR (Laboratorio Remoto) orientado a la enseñanza de las ciencias y la ingeniería de tal manera que resulte eficiente incluso cuando los elementos del laboratorio están distribuidos en el espacio y las comunicaciones (entre dispositivos y con respecto al usuario) aparece mediada por una red de telecomunicaciones (ya sea local o Internet)? Evidentemente, contestar este interrogante requiere un estudio de tecnologías y arquitecturas plausibles de utilizar en laboratorios remotos, tanto del ámbito educativo como industriales.

2 LOS LABORATORIOS REMOTOS Y LA TELEOPERACIÓN

La evolución de las redes de telecomunicaciones ha permitido su utilización en múltiples proyectos de ingeniería. Entre dichos proyectos se cuentan los que tienen que ver con la educación, por ejemplo los Laboratorios Remotos (LR). Por diversos motivos, económicos y de infraestructura entre otros, se está optando por implementar sistemas que permitan compartir experiencias de laboratorio a distancia, de tal manera que el alumno pueda "sentir" que maneja y controla el proceso bajo estudio "como si estuviera presente en el lugar", lo cual le permite adquirir habilidades en el manejo de dispositivos e instrumental real sin importar su ubicación geográfica (Callaghan y otros, 2003).

En la presente sección se analiza el estado del arte de los laboratorios remotos y el papel que desempeña la teleoperación en el funcionamiento de los mismos.

2.1 Los laboratorios remotos: el ayer y el hoy

Desde antaño la experimentación ha sido un pilar en la enseñanza de las ciencias. Sin embargo, el concepto tradicional de los laboratorios de prácticas, en el marco de las enseñanzas de tipo científico o tecnológico, plantea en la actualidad una serie de dificultades, especialmente de tres tipos:

- El elevado coste, tanto de adquisición como de mantenimiento, de los equipos necesarios.
- La (casi) imposibilidad de disponer en los centros educativos de salas de laboratorio con los recursos suficientes para la totalidad de los alumnos que cursan materias de este tipo.
- La no disponibilidad de los laboratorios en cualquier horario, dado que su uso presencial requiere, en general, la presencia del profesor. Esto lleva a una infrautilización de los recursos disponibles.

En virtud a estas problemáticas hoy en día se han comenzado a desarrollar laboratorios teleoperados o remotos a través de redes de telecomunicaciones. Una experiencia de uso de un Laboratorio Remoto puede ser vista como un usuario con un ordenador en un lugar distante que controla remotamente un experimento en una localización específica. Dispositivos de estas características vienen siendo utilizados desde hace más de 20 años en la industria, por ejemplo en el campo aeroespacial para el control de naves y telescopios espaciales. Las primeras aplicaciones de tele-manipulación a través de Internet tuvieron lugar hace poco más de una década principalmente en el campo de la robótica. En España, una Universidad pionera fue la Universidad de Valladolid, la cual comenzó a trabajar en este campo en 1998. Sin embargo se puede afirmar que la migración de este tipo de aplicaciones al ámbito de la educación, ha sido posterior y ha venido ligada, por un lado, al aumento del ancho de banda disponible para el acceso a Internet por parte de los usuarios, y, por otro lado, al abaratamiento tanto de las tarifas de conexión de alta velocidad como al de los propios ordenadores (González y otros, 2008).

Si bien no existen reglas axiomáticas se pueden, en virtud de la experiencia, establecer algunas pautas de diseño básicas a la hora de implementar un laboratorio remoto (González y otros, 2008):

- Los actuadores (ej.: motores) y sensores (ej.: termocuplas) deben poder ser accionados y leídos por un usuario remoto que se conecta al laboratorio a través de la Web (Internet).
- Debe existir una cámara que posibilite una vista general de la situación para que el operador remoto sienta que tiene el control del experimento, requisito fundamental para alcanzar una motivación óptima.
- El sistema debe contar con facilidad de acceso y sencillez de manejo.
- Las medidas tienen que ser rápidas y deben poderse visualizar gráficamente los resultados para que puedan ser validados y en caso contrario repetir la toma de datos.
- Además el usuario debe encontrar toda la información necesaria (manual de funcionamiento, guión de la práctica, etc.) en la misma aplicación.
- Por último el usuario ha de tener también la posibilidad de realizar consultas y presentar una memoria del experimento para que pueda ser corregida y convenientemente evaluada.

2.2 La teleoperación: definición y aplicaciones

Se entiende por "teleoperación" a la extensión de la percepción de una persona, la decisión de hacer, y la capacidad de manipulación a distancia (Hirchie y Buss, 2012). Los sistemas de teleoperación de hoy en día permiten la interacción con los ambientes a distancia y también se puede escalar la fuerza humana y el movimiento para conseguir capacidades de acción pese a restricciones de tamaño, fuerza, accesibilidad o incluso peligro para la vida. Las aplicaciones clásicas van desde la manipulación de residuos tóxicos/nucleares/inflamables hasta aplicaciones de rescate en entornos variados y cirugías de mínima invasión, pasando por la nanorobótica y aplicaciones orientadas a la educación y el entretenimiento (Hirchie y Buss, 2012).

En un sistema teleoperado, los comandos de control y las mediciones realizadas por sensores se transmiten utilizando diversos medios, entre ellos, microondas, radiofrecuencias y redes de computadoras. Dada la evolución de los sistemas de comunicaciones, los programas relacionados y los lenguajes de programación, hoy en día existen soluciones simples y económicas para el desarrollo de sistemas operados a distancia que utilicen Internet como medio de comunicación de alcance mundial (Liu, Chen y Meng, 2000).

Si bien se han desarrollado varios sistemas teleoperados sobre Internet (por ejemplo Golberg y Mascha, 1995; Xue, Yang y Meng, 2005; Lum, Rosen y otros, 2009), cada uno con sus particularidades, en general hay ciertas características que están presentes inherentemente en todos ellos y que nos dan una visión general del funcionamiento de un esquema de operación remoto; éstas pueden verse en la figura 1.

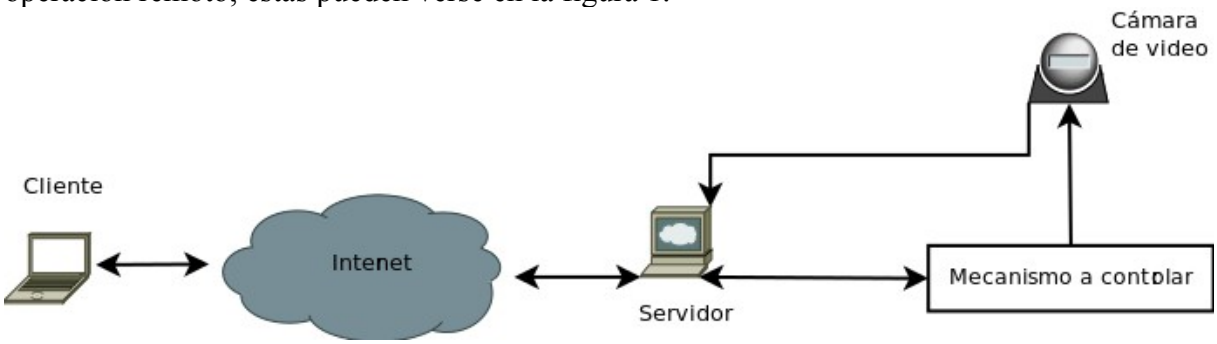

Figura 1: Arquitectura general de un sistema teleoperado

En la figura se observa a un usuario con un ordenador cliente que controla a través de Internet un experimento real en una localización remota. El experimento se realiza mediante diferentes mecanismos, motores y sensores que son controlados por un servidor Web y un interfaz gráfico. Una cámara permite seguir en tiempo real toda la experiencia (González y otros, 2008).

Como se puede apreciar, el sistema consta de dos grandes partes: cliente y servidor (Liu, Chen y Meng, 2000). El primero se consuma en la computadora del usuario, donde él puede monitorear todo el sistema y realizar las acciones a través de una interfaz gráfica de usuario (usualmente denominada GUI). Dicha interfaz debe poder manejar la comunicación bilateral que se establece entre el cliente y el sistema (red + sistema remoto), esto incluye presentar clara y convenientemente los datos recibidos, así como también enviar a la parte del servidor los comandos de control que elige el usuario. Dicha interfaz se prefiere amigable, con opciones de ayuda e íconos apropiados. También, existen nuevos enfoques donde incluso la interactividad y el video se mezclan en una composición muy novedosa y atrayente (Jailly, Preda y otros, 2011).

La parte del servidor, generalmente ubicada en el mismo sitio donde está el mecanismo o proceso a controlar, se ocupa de las políticas de acceso, servidor web y control local. Además,

este servidor también recibe el flujo de video de la cámara el cual contiene la información visual sobre el estado de situación del mecanismo bajo estudio y la retransmite a los clientes vía web.

3 TECNOLOGÍAS PARA EL DISEÑO DE LABORATORIOS REMOTOS

En varias industrias (entre ellas la robótica) los desarrollos conllevan la necesidad de integración de varias plataformas tecnológicas, junto con la utilización de múltiples lenguajes de programación y una exigencia de alta confiabilidad en el desempeño (Curry, 2004; McCafferty, 2010). En virtud a esto, se ve la necesidad de tener una herramienta o arquietectura de software que brinde a los desarrolladores la posibilidad de concentrarse en los objetivos finales de sus proyectos independizándose de los detalles del hardware de bajo nivel. Así, se llega al concepto de Middleware (sistemas de software de capa intermedia).

3.1 Middleware: Taxonomía y Definiciones

En el contexto de los sistemas distribuidos, la definición se puede enunciar así (Web Wikipedia): *software que provee servicios más allá de los proporcionados por el sistema operativo que permite a los diversos componentes de un sistema distribuido comunicar y gestionar los datos*.

Ésta plataforma simplifica el trabajo de los programadores en la compleja tarea de generar las conexiones que son necesarias en los sistemas distribuidos (Web Wikipedia). Los frameworks de capa intermedia están diseñados para enmascarar algunos tipos de heterogeneidad con los que tienen que lidiar los programadores de sistemas distribuidos, entre los cuales se cuentan: detalles sobre redes, sistemas operativos, hardware, e incluso lenguajes de programación (Bishop y Krane, 2003). En la figura 2 se muestra un gráfico muy claro de la utilización del middleware en el contexto de los sistemas distribuidos.

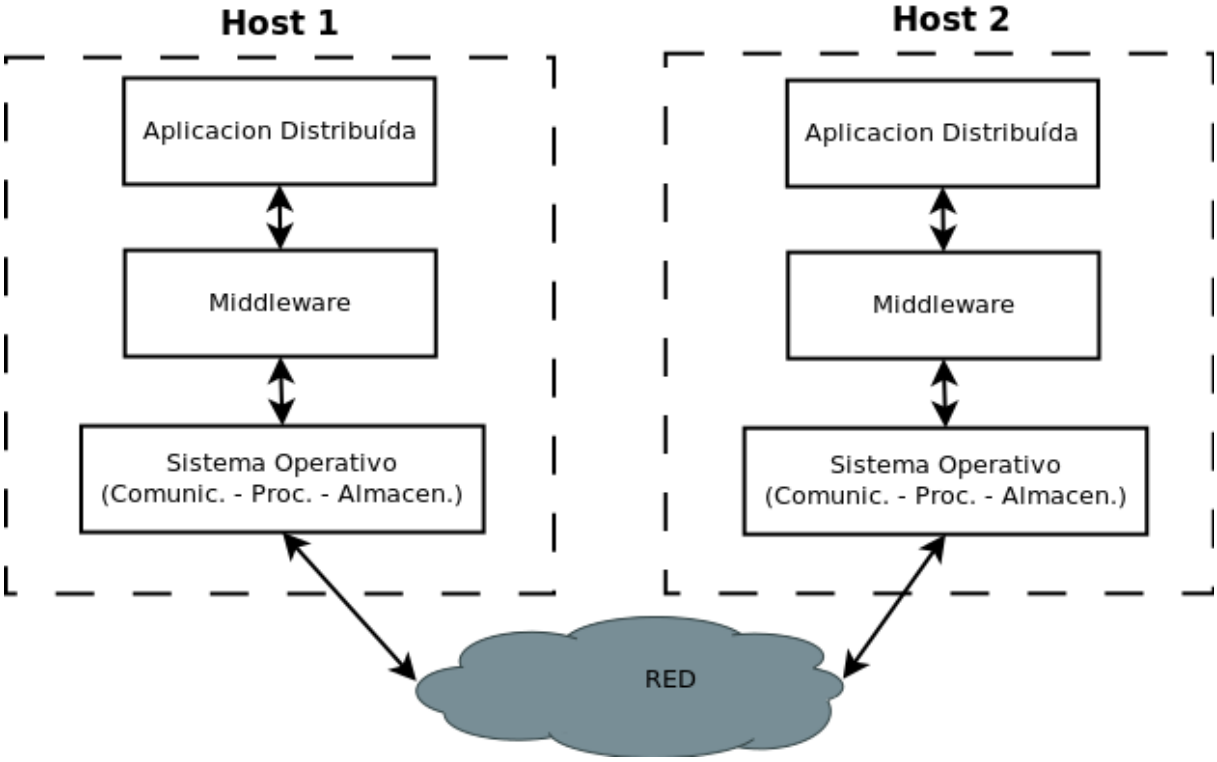

Figura 2: Middleware y sistemas distribuidos(Web Wikipedia)

Los software de capa intermedia pueden clasificarse en dos grandes categorías: integración

y aplicación. Los primeros incluyen la capacidad de unirse con sistemas heterogéneos y generalmente poseen diferentes protocolos de comunicación. Por otro lado, los segundos son aquellos ajustados específicamente para alguna aplicación. En virtud a que en los LR existe una gran variedad de elementos a conectar, algunos distribuidos y multiplataforma, se ve la necesidad de un *middleware de integración*. Dentro de éste grupo se hace incapié en los MOM (Middleware Orientados a Mensajes) dado que, como se explica en la siguiente subsección, tienen una importante característica de funcionamiento asíncrono.

3.2 Capa Intermedia orientada a Mensajes (MOM)

Procurando satisfacer las demandas de conectividad heterogénea que presentan varias industrias (entre ellas las relacionadas con robótica) ha surgido un mecanismo llamado Capa Intermedia Orientada a Mensajes (MOM, por sus siglas en inglés), el cual provee un método de comunicación transparente entre entidades de software distribuidas. MOM puede definirse como cualquier infraestructura de capa intermedia que provee capacidades de mensajería (Curry, 2004).

Un cliente de un sistema de MOM puede enviar y recibir mensajes hacia y desde otros clientes del sistema de mensajería, considerándose a todos como iguales (comunicación entre pares). Cada cliente se conecta a uno o más servidores que actúan como intermediario en la transacción de mensajes. Este tipo de plataforma permite la creación de sistemas con cohesividad flexible, esto significa que los cambios en una parte de la estructura se pueden llevar a cabo sin la necesidad de modificar otras partes de la misma.

Además, los sistemas MOM proporcionan comunicación distribuida sobre la base del modelo de interacción asíncrona. Esto implica un modelo sin bloqueo donde los participantes del sistema no están obligados a bloquear y esperar respuesta tras el envío de un mensaje, sino que *se les permite continuar* con el proceso interno (de ahí que se diga que no hay bloqueo) hasta que, al llegar la respuesta deseada, se atiende ésta.

En la figura 3 puede notarse que la arquitectura de estos sistemas están pensadas para ofrecer una comunicación entre procesos orientada a la prestación de servicios. Además, dado que MOM inserta una capa (que sirve de intermediario) entre las partes que intercambian mensajes, se pierde necesidad de acoplamiento, lo que resulta en un sistema más cohesivo y donde no es necesario hacer tantas adaptaciones entre fuentes y destinos de mensajes. Esto es una ventaja muy importante.

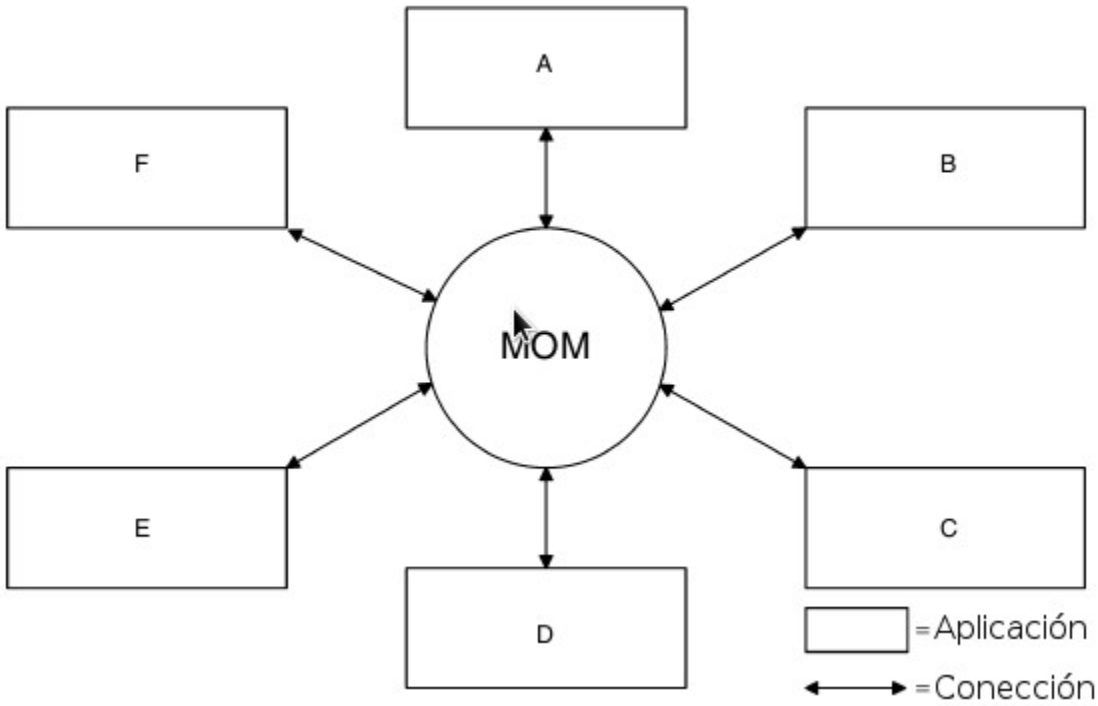

Figura 3: Esquema de sistema MOM (Curry, 2004)

También, entre las características generales de este tipo de sistemas puede decirse que resultan muy confiables, frecuentemente muy configurables y cómodamente escalables. Además, los MOM son capaces de garantizar que los mensajes se entregarán y lo harán sólo una vez (sin repeticiones o "ecos") al destinatario.

3.3 Desempeño de los sistemas basados en mensajes

Para entender la ventaja fundamental de los MOM, hay que entender la diferencia entre un sistema síncrono y uno asíncrono; para esto resulta de utilidad la figura 4.

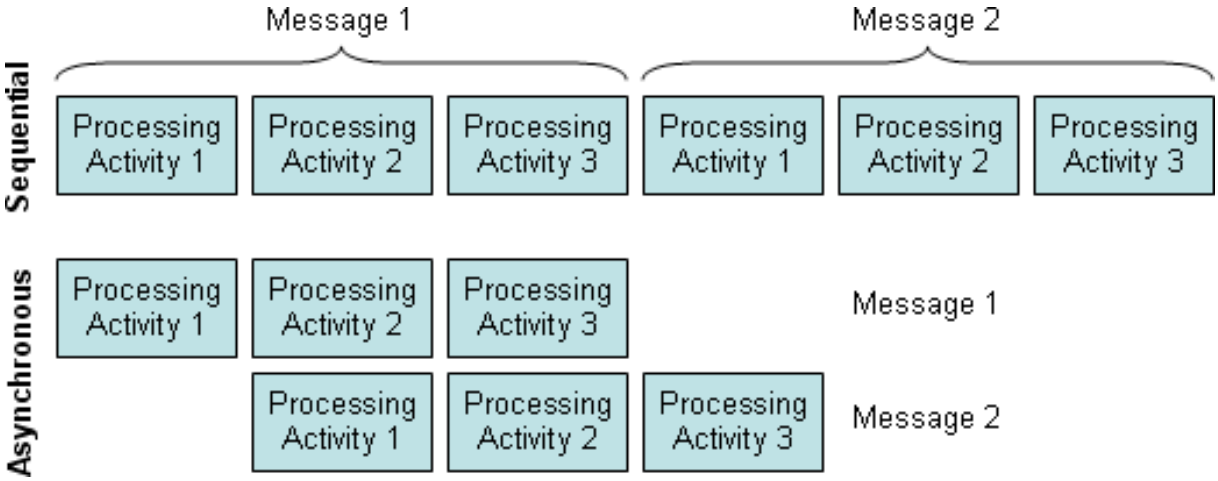

Figura 4: Comparación de manejo de tareas entre un sistema síncrono y uno asíncrono (McCafferty, 2010)

Puede notarse en la antedicha figura que el proceso secuencial del procesamiento

sincrónico conlleva la demora de que antes de pasar a procesar un mensaje es requisito que se haya terminado de trabajar con el anterior. En cambio, en el modelo asincrónico se puede empezar a procesar el siguiente mensaje cuando el primero haya sido procesado en parte. Esto compensa la sobrecarga adicional impartida por esta estructura de mensajería.

Por otro lado, se puede decir que una fortaleza de los Middleware orientados a mensajes radica en su capacidad para almacenar, enrutar y transformar los mensajes mientras son transportados desde los emisores hasta los receptores. A su vez, su particular desventaja radica en la necesidad de un agente para transferencia de mensajes (broker) el cual significa un componente más en el sistema y, consecuentemente, más complejidad y gasto de mantenimiento. Además, cabe aclarar que dado el funcionamiento intrínsecamente asincrónico de los MOMs estos pueden no ajustarse bien en algunas situaciones donde se exija respuestas en tiempo real.

Aún así, para la aplicación en el contexto de los laboratorios remotos *es posible utilizar* MOMs. Algunos trabajos que establecen este hecho y muestran resultados de aplicación son Tosco, Corteggiano y Broll, 2012; Tosco, Corteggiano y Broll 2013; Web Remote-Lab; Osentoski y otros, 2012; Fayolle, Gravier y Jailly, 2010. Puede apreciarse en ellos que la implementación implica retardos tolerables menores a 25[ms], lo que según (Guinaldo, Sanchez y Dormido, 2010) corresponde a una cota para laboratorios remotos.

De la amplia gama de sistemas de Capa Intermedia Orientada a Mensajes disponibles, se optó por elegir uno en particular: El Sistema Operativo Robot (ROS, por sus siglas en Inglés).

4 LA CAPA INTERMEDIA "ROBOT OPERATION SYSTEM" (ROS)

En el presente trabajo se ha escogido utilizar el Robot Operation System (ROS) dentro de un sistema de teleoperación en el contexto de los laboratorios remotos de nueva generación para evaluar su desempeño. Las razones de esta elección se fundamentan en tres aspectos: las prestaciones de esta arquitectura de software, la calidad de documentación y repositorios, y la buena compatibilidad con el lenguaje de programación Python.

4.1 Aspectos generales

ROS es un meta-sistema operativo de código abierto pensado para robótica que corre sobre plataformas basadas en UNIX. Ofrece los servicios que se esperarían de un sistema operativo, incluida la abstracción de hardware de bajo nivel de control del dispositivo, la aplicación de la funcionalidad de uso común, el paso de mensajes entre los procesos y la gestión de paquetes. También proporciona herramientas y bibliotecas para la obtención, construcción, escritura y ejecución de código en varios equipos (Quigley y otros, 2009). Además, la documentación sobre éste es muy amplia y completa.

La filosofía de ROS se puede resumir en pocos items:

- Punto a Punto
- Basado en herramientas
- Multilenguaje (actualmente son soportados cuatro lenguajes: C++, Python, Octave y LISP.)
- Modular (microkernel con funciones específicas + módulos con librerías y drivers)
- Gratis y de código abierto (licencia BSD)

4.2 Funcionamiento

En ROS se considera que todo proceso, o parte de un sistema que realiza cálculos, es un *Nodo*. Además hay un *Maestro*, el cual es un servicio de nombres para ROS que ayuda a los nodos a contactar con sus pares y obtener información útil. Entre los nodos puede compartirse

información a través de dos paradigmas: *Cliente-Servidor* (esto sería tipo Petición-Respuesta, modelo síncrono, o sea bloqueante, basado en XML-RPC) y *Publicador-Subscriptor*. Ambos paradigmas están muy bien resumidos en la figura 5.

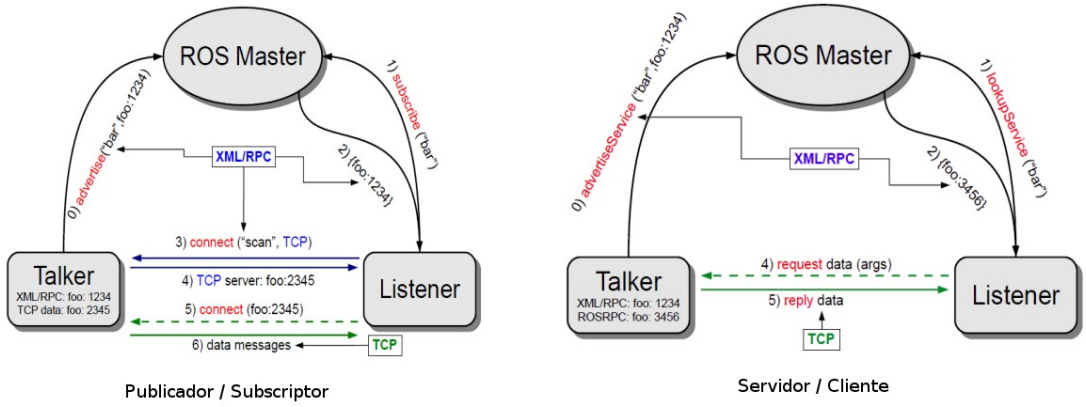

Figura 5: Paradigmas de comunicación en ROS (Ros-Tutorial, 2014)

En el paradigma Publicador-Subscriptor, un nodo publica datos a un canal de comunicación llamado *Tópico* y otro se subscribe al mismo para acceder a los datos. Pueden haber varios publicadores y subscriptores. Es un modelo de funcionamiento flexible (asincrónico). Aquí el maestro actúa sólo como un servicio de nombres almacenando la información de registro de los temas (tópicos) y servicios de los nodos ROS. Los nodos se comunican con éste para reportar su información de registro. A medida que los nodos se comunican con el maestro, pueden recibir información sobre otros nodos inscriptos y hacer las conexiones, según corresponda. También éstos recibirán notificación de los cambios en la información de registro, así pueden crear dinámicamente las conexiones con los nuevos nodos. *Los nodos se conectan a otros nodos directamente, el Maestro sólo proporciona información de búsqueda*.

4.3 Algunas ventajas y desventajas de ROS

Es de interés que se ha realizado una comparación entre varias plataformas de software y se han discriminado las virtudes y algunos puntos débiles de la plataforma escogida.

Entre las virtudes se destacan: orientado a mensajes, multisistema operativo, código libre, permite trabajar en entornos distribuidos, posee simulador y está bien documentado. Entre los puntos débiles se tiene que no posee seguridad implementada (puede salvarse este escollo utilizando conexiones seguras "ssh") y que no es el middleware ideal para aplicaciones exigentes en tiempo real (existen otras como OROCOS diseñadas para tal fin (Elkady y Sobh, 2012)).

5 USO DE SISTEMAS EMBEBIDOS COMO NODOS DE ADQUISICIÓN Y PROCESAMIENTO

Habiendo explicado las cuestiones relacionadas con software, se procede a continuación a explicar el papel de los sistemas embebidos como nodos de adquisición en el contexto de los laboratorios remotos. Una definición concisa de sistema embebido o empotrado podría ser la siguiente (Web Wikipedia): *Un sistema embebido (anglicismo "embedded") o empotrado es*

un sistema de computación diseñado para realizar una o algunas pocas funciones dedicadas, frecuentemente en un sistema de computación en tiempo real.

Como una aplicación derivada se puede hablar de "Linux embebido", el cual se refiere al uso del núcleo Linux en un sistema embebido, como por ejemplo PDA, teléfonos móviles, robots, enrutadores, servidores, dispositivos electrónicos y aplicaciones industriales con microcontroladores y microprocesadores. Si bien existen otros sistemas operativos empotrados, el linux presenta ventajas significativas por ocupar un tamaño reducido y por ser de código libre con amplia utilización en el campo.

Típicamente, el corazón de este tipo de sistemas es un microcontrolador, aunque los datos también pueden ser procesados por un DSP, una FPGA, un microprocesador, etc., y su diseño está optimizado para reducir su tamaño y su costo, aumentar su confiabilidad y mejorar su desempeño. El diseño de sistemas embebidos es un motor clave de la industria y del desarrollo tecnológico, y es un campo que en los últimos años ha crecido notablemente en la Argentina (Web oficial SASE).

¿Qué ventajas trae aparejado el utilizar este tipo de sistemas? Por un lado, la especificidad de la aplicación o sistema operativo. Sí, muchas veces el programa se realiza en el ensamblador propio del microprocesador, con lo cual se evitan redundancias de código o capacidades ociosas por ser una implementación ad-hoc. Además, la reducción en costos de hardware es significativa.

También, si se utiliza un sistema operativo (S.O.) embebido ello permite utilizar la complejidad de los programas diseñados para PCs de escritorio pero con la capacidad y tamaños mínimos de los ordenadores embebidos. También, dicho tipo de S.O. permite aprovechar herramientas de desarrollo de software conocidas y dominadas por muchos programadores en el mundo.

Huelga decir que el consumo en potencia eléctrica es, en general, mínima y los tamaños de los dispositivos son pequeños.

5.1 BeagleBone, una buena plataforma de trabajo

Todas las características mencionadas hicieron que la utilización de computadores embebidos fuera una opción atrayente en la realización de este trabajo. En particular se escogió un hardware en particular (que está basado en un microprocesador ARM) llamado Beaglebone A5. En la figura 6 se muestra una imagen representativa de éste.

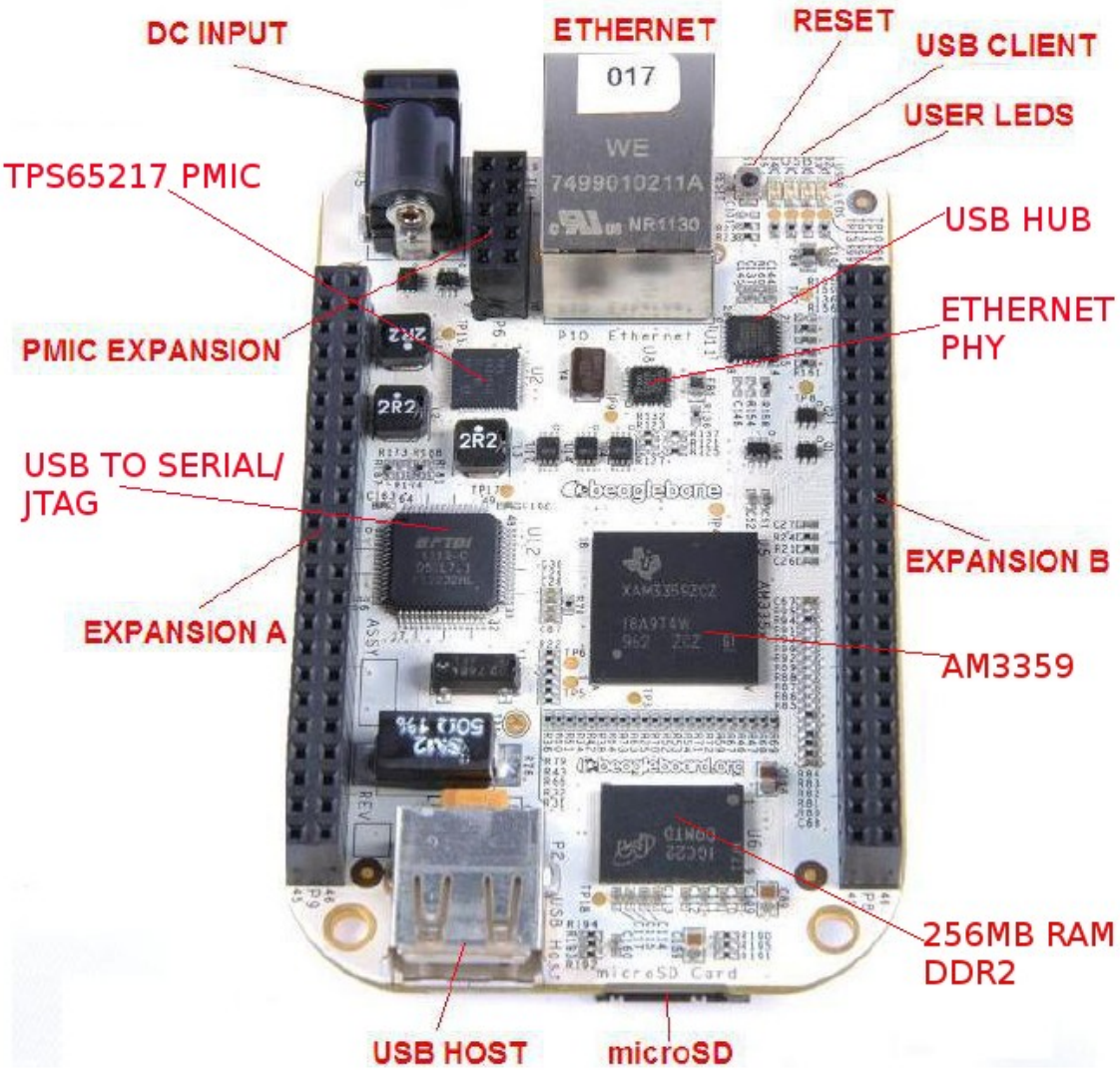

Figura 6: Beaglebone A5

BeagleBone es un ordenador pequeño pero con el que también es posible controlar hardware externo a través de sus pines. Su procesador es un ARM cortex A8 que funciona a unos 700mhz (más adelante se muestran más datos) y cuenta con 256MB de RAM.

Con estas prestaciones es posible ejecutar un sistema operativo como el existente en un ordenador, siempre y cuando exista una versión compilada para procesador ARM; existen distribuciones de linux para arm. Se utilizó para las pruebas realizadas Linux Ubuntu 12.04.3 LTS (GNU/Linux 3.8.13-bone27 armv7l).

En cuanto a su tamaño puede decirse que es realmente pequeño, unos 9cm de largo y 5.5 de ancho, y cuenta con un lector de tarjetas microsd (que hace las veces de "disco duro" para almacenar el SO y los datos), un host USB (para conectar dispositivos), un conector RJ45 y un micro USB para vincularlo al PC.

Como apreciarse en la figura 6, el microprocesador es un AM3359.Dicho procesador tiene una frecuencia variable según la tensión de alimentación. Así, cuando es alimentado por el USB su velocidad de reloj es 500MHZ, mientras que si se la alimenta desde la entrada DC sube a 700MHZ.

Con respecto a su memoria RAM es de 256 MB DDR2 de 400MHZ. Es suficiente para correr un sistema operativo modo consola de manera fluida. La placa no tiene salida de video.

Con respecto al consumo puede decirse que en el peor de los casos éste ronda los 0,5 [A]. Sin embargo, si se le va a acoplar dispositivos al puerto USB se recomienda el uso de una fuente externa de 2 [A].

¿Por qué la elección de BeagleBone? En resumidas cuentas puede decirse que se eligió esta placa por cinco razones:

- Cumplía con los requerimientos mínimos que se necesitaban para el presente trabajo
- Era una minicomputadora con bancos de expansión y no sólo un microcontrolador con interfaces (como otras placas de desarrollo; por ej. Arduino)
- El costo era accesible.
- Se conseguía en el país y en tiempos razonables.
- La documentación es muy buena y hay bastante gente trabajando con ella.

5.2 Uso de la BeagleBone como nodo ROS

Utilizando comandos ROS apropiados se creó la estructura de archivos necesaria (*paquete* ROS) para soportar un nodo, al que se nomencló "planta.py" (".py" es la extensión característica de un programa hecho en lenguaje Python) el cual emula, mediante la medición de tensión sobre un circuito RC, el llenado de un tanque de agua (en la siguiente sección se dan más detalles). En la figura 7 se muestra la porción del código que resulta de interés en el uso de ROS (<...> indica que ahí va código que no es pertinente y por tanto no se muestra).

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
    nodo_planta.py - Nodo que simula planta (Circuito RC)

    Copyright (C) 2013 SJTosco.

    <...>

"""

import roslib; roslib.load_manifest('rc_sim')
import rospy
from std_msgs.msg import Float64, String
import <...>

class Planta:
    def __init__(self):
        rospy.init_node('nodo_planta', anonymous=True)

        rospy.on_shutdown(self.cleanup)

        <...>
        # We don't have to run the script very fast
        self.rate = rospy.get_param("rr", 5)
        r = rospy.Rate(self.rate)

        # Publish Vout message to the y topic
        self.pub_y = rospy.Publisher('y',Float64) #Publicador y

        # Subscribe to the u topic to receive pwm commands.
        rospy.Subscriber('u', Float64, self.callback_u) #Subscriptor u
        <...>

        # We have to keep publishing the cmd_vel message if we want the robot to keep moving.
        while not rospy.is_shutdown():
            <...>
            self.pub_y.publish(Float64(V_Cap))
            r.sleep()

    <...>

    def callback_u(self, data):
        <...>


if __name__=="__main__":
    try:
        Planta()
        rospy.spin()
    except rospy.ROSInterruptException:
        rospy.loginfo("Planta (rc_sim) terminated!")
```

Figura 7: Código "planta.py"

Puede notarse en la antedicha figura que, tras el comentario inicial (rótulo) se hacen ciertas

declaraciones para importar librerías, entre ellas "rospy" la cual permite usar ROS desde Python y también, en la línea 14 se importan los tipos de mensajes estandar Float64 y String, los cuales son utilizados para definir qué clase de información circulará por los tópicos de comunicación.

Para quienes conocen la Programación Orientada a Objetos (POO) resulta evidente que lo que se define a continuación es una clase llamada Planta la cual tiene varios métodos, el primero de los cuales es "__init__(self)" que se ejecuta al instanciarse la clase. Reviste interés especial la línea 19 donde se inicializa (o crea) el nodo llamado "nodo_planta" con la opción "anonymous=True" que implica correr los nodos sin tomar en cuenta los nombres.

A continuación hay varias configuraciones invocando la funciones de "rospy" y después, en las líneas 29 y 32 se crean los objetos publicadores y subscriptores, respectivamente.

Puede notarse que ambos manejan un tipo de dato Float64 (punto flotante, alta precisión) y en el caso del subscriptor puede verse que se define el nombre de la función que se ejecutará al recibir un dato desde el tópico (tema) al que se está inscripto, llamada "callback_u".

También resulta evidente que hay dos tópicos distintos que resultan de interés: "y" para el publicador y "u" para el subscriptor.

Por otro lado, en la línea 38 puede verse cómo se invoca el método "publish" del objeto publicador para enviar un dato hacia el tópico de interés.

Finalmente, desde la línea 48 puede notarse la instanciación de la clase "Planta" y el comando "rospy.spin()" el cual pone en marcha un ciclo especial en ROS que implica quedar a la espera de que el servicio o nodo deje de estar activo (se apague).

5.3 Nodo servidor MJPEG

En el BB es posible instalar el servidor MJPG-Streamer, el cual corre en modo consola y sirve para difundir vía una red IP fotogramas con compresión JPEG desde una webcam hasta un visualizador como por ejemplo un explorador web, un reproductor como el VLC o cualquier dispositivo móvil con navegador web (Web oficial MJPG-Streamer).

Una vez instalado, es posible crear un nodo ROS llamado "nodo_camara" contenido en el archivo "nodo_camara.sh" (el cual tiene los permisos de ejecución en el S.O.). Este archivo posee una configuración determinada del servidor de video y su contenido se muestra en la figura 8.

Figura 8: Código "planta.py"

Puede verse en la figura 8 que mjpeg_streamer funciona como un "pegamento" entre complementos (plug-ins) que realizan distintas tareas. Por ejemplo, arriba puede apreciarse que tras indicar el parámetro "-i" (entrada) se invoca el plug-in "input_uvc.so" (que sirve para conectar con cámaras compatibles con V4L2) con indicación de la ubicación de la cámara en el sistema linux y una definición del tamaño de fotograma. Además, se configura tras el parámetro "-o" (salida) el plug-in "output_http.so" (éste es un webserver http totalmente funcional) con indicación de usar el puerto 8080 y servir la web en la carpeta /usr/local/www. El signo "&" es para que la terminal no se bloquee y lance el proceso por separado.

¿Conviene utilizar MJPG-Streamer? Esta herramienta no depende de ROS y por tanto, una ventaja al utilizarla es que una falla en la emisión de video no afecta al sistema del laboratorio remoto en sí. Además, para tamaños pequeños (por ejemplo 160x120) la carga en la red no es significativa para una conexión a Internet estándar. También, hay que reconocer que la compatibilidad de los navegadores para mostrar flujo MJPEG en HTML5 es mucho mayor que si se usara uno con video comprimido en otros estándares (Por ejemplo H.264). También conviene destacar que MJPG-Streamer es muy fácil de manejar e instalar en sistemas linux como los utilizados para este trabajo e incluso, sin ser parte de ROS, es compatible con éste en algunos aspectos (por ejemplo al poder lanzarse ambos desde un sólo archivo de configuración). Es menester mencionar que esta herramienta fue diseñada para sistemas con pocas prestaciones de CPU y RAM. Como desventaja puede mencionarse que para grandes tamaños de fotogramas el requerimiento de ancho de banda podría ser un escollo.

También, es posible realizar un nodo que procese los fotogramas capturados mediante técnicas digitales y brinde información de interés sobre el evento físico mostrado. Por ejemplo, la figura 9 muestra el resultado de correr un nodo de procesado de imágenes que calcule un altura a partir de un flujo de fotogramas (se utiliza como técnica base la umbralización).

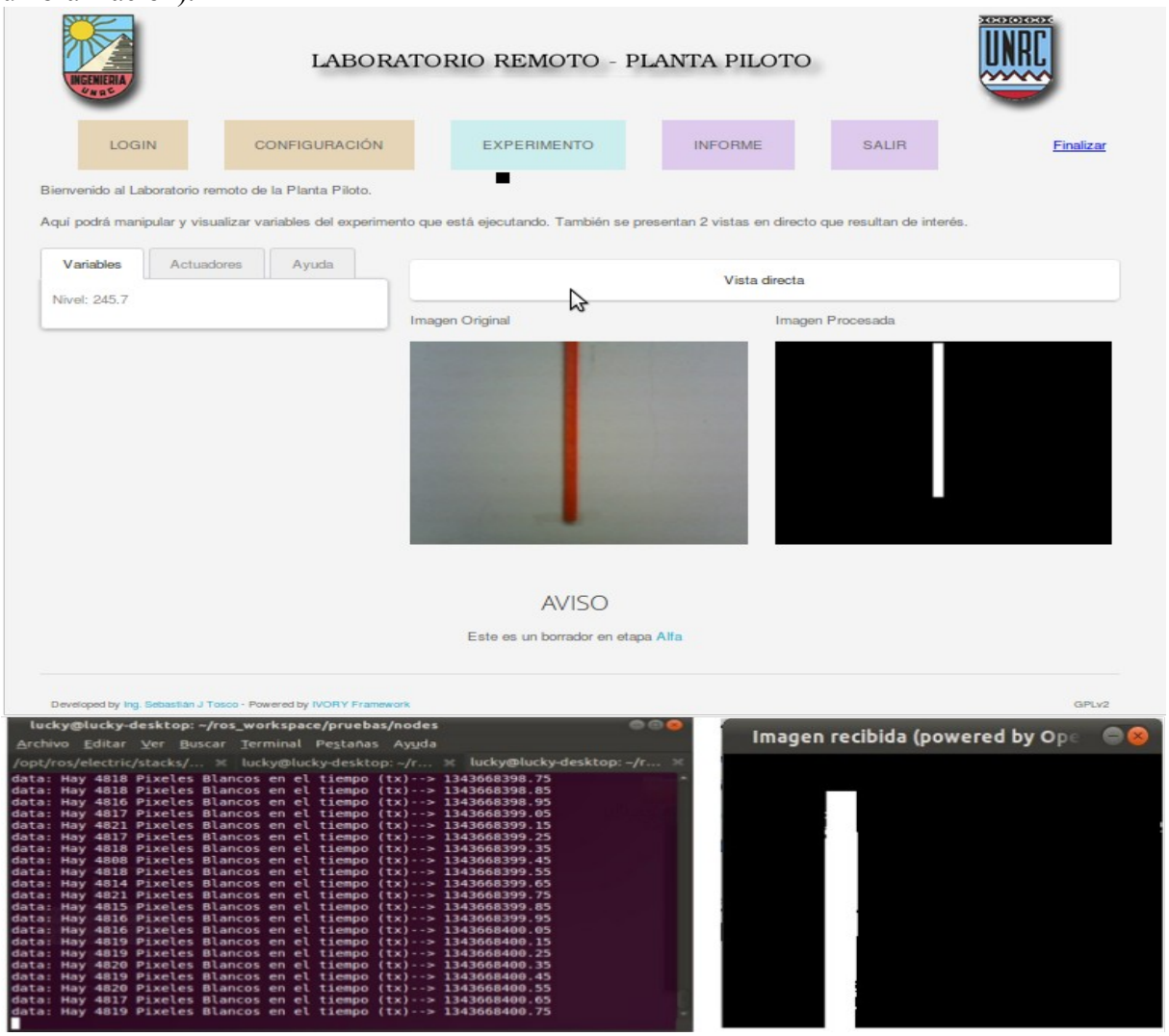

Figura 9: Nodo ejemplo de procesamiento de imágenes en funcionamiento (entorno web versión alfa)

Como se ha apreciado en esta sección, los sistemas embebidos poseen ventajas interesantes

que pueden ser muy útiles a la hora de diseñar un laboratorio remoto. Entre ellas figuran: su reducido coste, su modesto tamaño, el consumo notablemente bajo de potencia y la versatilidad que se otorga al programador permitiendo, en muchos casos, embeber un sistema operativo completo. El equipamiento escogido, BeagleBone, demostró comportarse muy bien en diversos ensayos.

En la siguiente sección se describirá con detalle el prototipo de laboratorio remoto que se ha implementado, puntualizando los aspectos que resultaron de particular interés en el desarrollo del presente trabajo.

6 IMPLEMENTACIÓN DE UN CASO DE ESTUDIO PARA LA ARQUITECTURA PROPUESTA

Con el objeto de probar la eficacia de la arquitectura propuesta (ROS) se construyó un escenario de prueba el cual permite relevar el comportamiento del sistema y evaluar las posibilidades de implementación en laboratorios remotos más complejos orientados a la enseñanza de la ingeniería. Su estructura general se muestra en la figura 10.

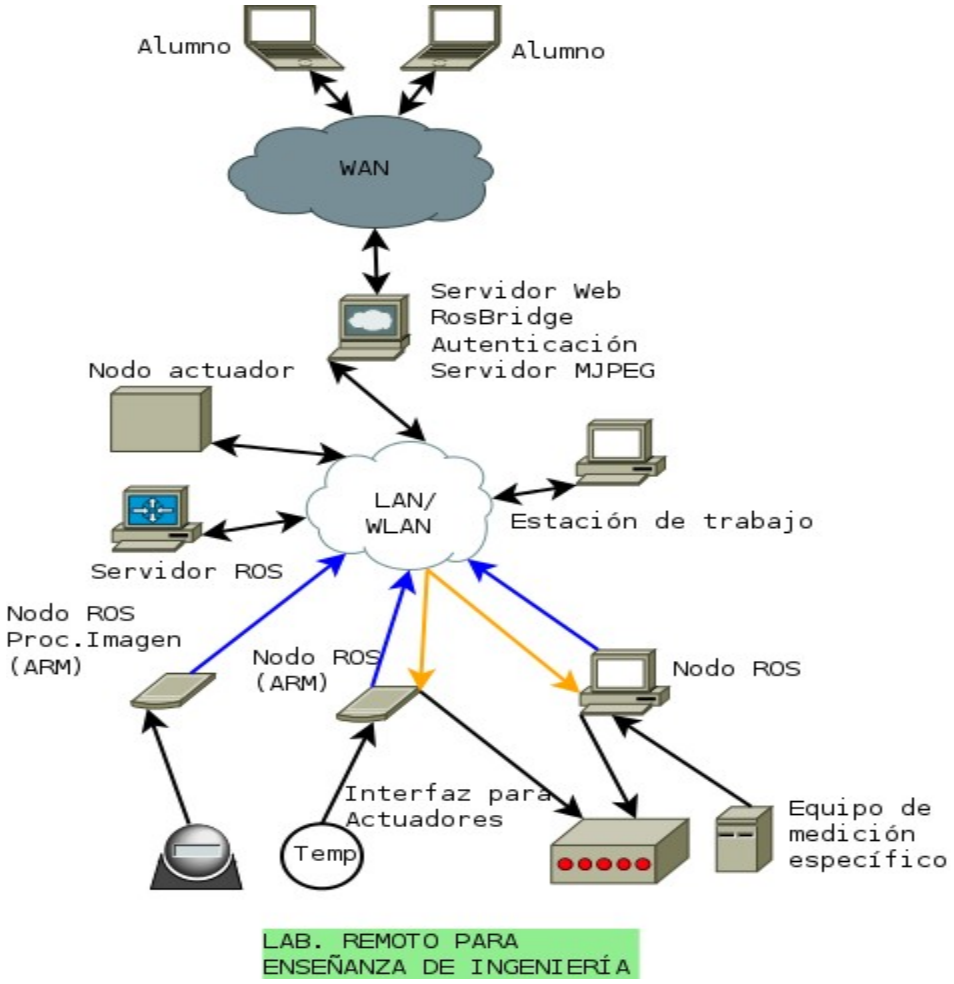

Figura 10: Caso de estudio: diagrama general

Como puede apreciarse en la figura 10, el sistema posibilita a los alumnos conectarse a través de Internet (puede incluirse validación de usuario) accediendo al servidor principal, el cual posee varias funciones, entre ellas:

- Gestor de conexiones, usuarios y autenticación (en caso de implementarse)

- Servidor web
- Servidor ROSBRIDGE (interface entre ROS y la Página Web vía lenguaje Javascript)
- Servidor MJPEG

Resulta especialmente atractivo que la presentación al alumno se hace como una página web, desde donde puede ver, no solo el estado de los sensores y actuadores, sino también controlar a distancia (manual o automáticamente) el experimento. Por tanto, la computadora que se conecte como usuario del sistema sólo necesita disponer de un explorador web actualizado (soporte HTML5 + CSS3 + Javascript).

Puede decirse que el servidor principal sirve de interfaz entre ambos lados de la arquitectura: lado remoto y el laboratorio en planta. Por tanto, una de sus funciones puede ser la gestión de usuarios, de tal manera que sólo accedan los usuarios que están autorizados (por ejemplo los alumnos y docentes de la unidad académica responsable del laboratorio remoto) y a su vez que no accedan dos usuarios en simultáneo, para no generar conflictos. Esta parte del escenario (la autenticación), si bien importante es posible implementarla a futuro.

Por otro lado, este servidor también sirve la página web que el lado cliente consulta de forma remota para visualizar y comandar el laboratorio remoto. Dicha web está programada en HTML5 y se utiliza una interfaz que, vía el lenguaje javascript, proporciona la librería ROSBRIDGE. La mencionada librería funciona como puente entre el usuario y la arquitectura ROS subyacente, de tal manera que el primero pueda visualizar y controlar a través de controles web (widgets) los sensores conectados a los nodos ROS. Por otra parte, el servidor MJPEG sirve para enviar un flujo de video comprimido (JPEG cada fotograma) al cliente sobre una vista general de la situación del laboratorio; esto puede resultar orientativo al usuario. Es interesante destacar que este sistema permite al usuario configurar su propia experiencia de laboratorio mediante elegir entre un modo de control automático y manual. En caso de ser automático, se puede configurar un nodo actuador que implemente en local un modelo de control planteado por alumno para manejar el proceso (dentro del esquema PID). Así, vía web, el alumno ve el resultado de su intervención en la respuesta del sistema.

Hasta el momento, y por simplicidad, todas las funciones del "servidor principal" se supusieron en un solo ordenador, pero en realidad podrían ser llevadas a cabo por varias computadoras. De hecho en la implementación que corresponde a las mediciones de este trabajo se implementó de la manera que se muestra en la figura 11, donde se ve un servidor WEB externo vinculado por una WAN (Internet) al sistema.

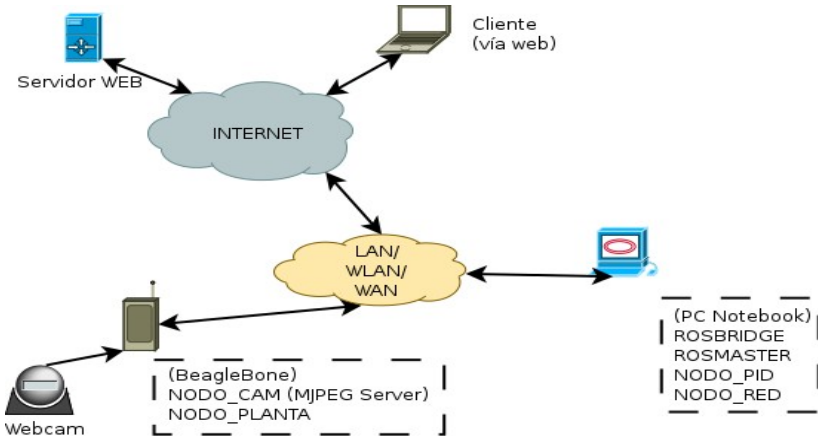

Figura 11: "Servidor principal" distribuido en tres dispositivos

Por lo que puede apreciarse hasta aquí, es evidente que los nodos ROS son una parte

importante y muy activa de la arquitectura, pues están vinculados con sendos sensores y actuadores.

6.1 Nodos creados en ROS para el caso de estudio

Tras consultar con docentes de trayectoria en el campo de las experiencias de laboratorio, se pensó en un caso sumamente simple: controlar el llenado de un tanque de agua. Este escenario se tomó como una buena base dado que los docentes asesores veían la posibilidad de ir ampliando el montaje del experimento mediante otros tanques y válvulas para adecuarlo a consignas complejas en materias afines a la Ingeniería Química.

El sistema de prueba armado tiene 4 nodos:

- Nodo de video (cámara) –> nodo_cam
- Nodo planta (simula RC) –> nodo_planta (Del cual se habló en la sección anterior)
- Nodo controlador (PID) –> nodo_pid (Realiza control Proporcional-Derivativo-Integral)
- Nodo red (NETEM) –> nodo_red (Emula características de Red WAN, como retardo y variación del retardo)
- Nodo de graficación (opcional - se eliminó en la versión final)

Estos nodos son lanzados desde un archivo central llamado rc_sim.roslaunch, el cual tiene estructura XML; puede apreciarse su contenido en la figura 12. Este archivo es muy importante pues permite con un solo comando lanzar todo el sistema, incluso al estar distribuido en múltiples computadoras.

```
  GNU nano 2.2.6                        File: launch/rc_sim.launch

<launch>
        <!--PARAMETROS-->
        <rosparam param="rr">100</rosparam>
        <rosparam param="sp">50</rosparam>
        <rosparam param="Kp">0.075</rosparam>
        <rosparam param="Ki">50</rosparam>
        <rosparam param="Kd">0</rosparam>

        <!--BB-->
        <machine name="bb" address="arm" env-loader="/opt/ros/groovy/env.sh" user="root" password="root" timeout="120"/>


        <node machine="bb" pkg="rc_sim" type="nodo_camara.sh" name="nodo_cam" args="" output="screen">
        </node>

        <node machine="bb" pkg="rc_sim" type="nodo_planta.py" name="nodo_planta" output="screen">
        </node>

        <!--PC-->
        <machine name="pc2" address="192.168.1.250" env-loader="/opt/ros/groovy/env.sh" user="root" password="danilo" />
        <include file="$(find rosbridge_server)/launch/rosbridge_websocket.launch">
           <arg name="port" value="9090"/>
        </include>

        <node machine="pc2" pkg="rc_sim" type="nodo_pid.py" name="nodo_pid" output="screen">
        </node>
        <node machine="pc2" pkg="rc_sim" type="nodo_red.py" name="nodo_red1" output="screen">
        </node>
</launch>

^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

Figura 12: Archivo “rc_sim.roslaunch”

En cuanto a la estructura interna del archivo “rc_sim.launch” se comenta que allí se aprovecha a establecer los parámetros centrales del sistema, se declaran los nodos a poner en funcionamiento y se identifican los terminales en donde están almacenados dichos nodos.

También se activan los nodos servidores, como son el ROS_CORE y el ROS_BRIDGE en el mismo archivo.

6.2 Algunos resultados interesantes

En las sendas pruebas que se realizaron pudieron establecerse resultados interesantes en cuanto a desempeño y funcionamiento de la arquitectura propuesta para implementar laboratorios remotos. A continuación se describen algunos resultados obtenidos.

6.2.1 Interfaz gráfica html

Las pruebas de uso de la página web demuestran que el diseño HTML5 responsivo es muy apropiado para la aplicación, dado que permite acomodar la estructura de la web a diversos dispositivos y tamaños de pantalla. Por ejemplo, la figura 13 muestra cómo se visualiza la interfaz web cliente en una pantalla de notebook 14" y en una tablet 4.3". En ambos casos es factible para el usuario usar el sistema con facilidad. Además, se dispone de una breve ayuda que permite visualizar el escenario.

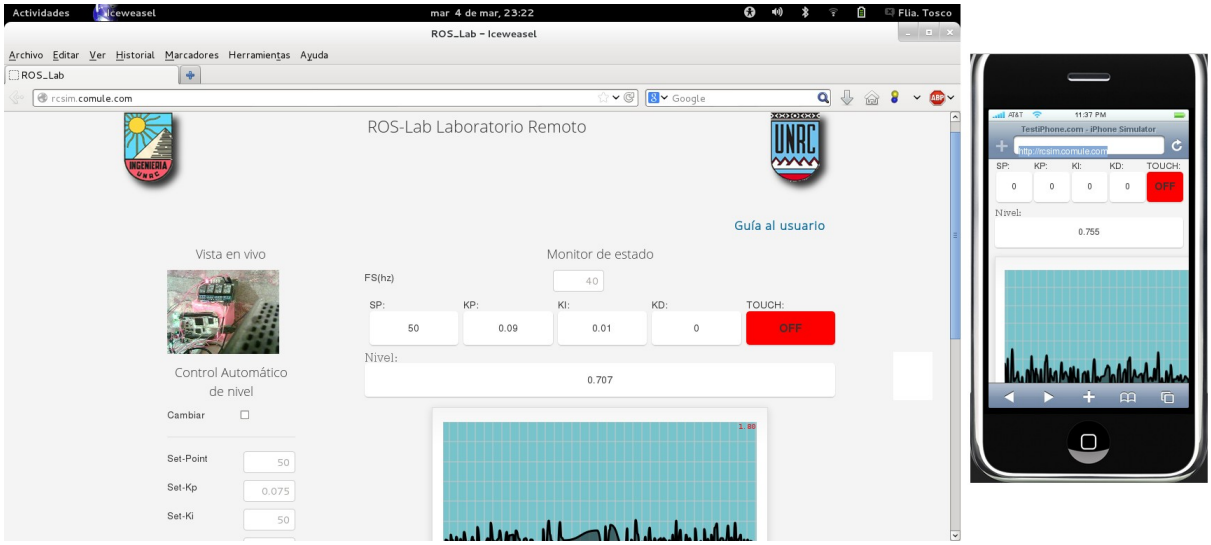

Figura 13: Visualización de la interfaz web en 2 dispositivos distintos

6.2.2 Tráfico de datos en el sistema

En la figura 14 puede notarse una captura de unos 2 minutos de duración del sistema propuesto en funcionamiento. Como puede apreciarse, el tráfico MJPEG de video sigue siendo lo mismo que se obtuvo con anterioridad (unos 150Kbps). El tráfico caracterizado por ROS (pto 9090 correspondiente al ROS_BRIDGE) se estima en unos 180 Kbps y el tráfico total del sistema ronda los 300 Kbps, los cuales no resultan ser privativos en una red moderna.

Figura 14: Tráfico en el sistema

6.2.3 Influencia de los parámetros de control

Como es de esperarse, la configuración de los parámetros de control utilizados en el sistema tienen gran influencia en el comportamiento del mismo. Así, por ejemplo, se ha podido establecer que el aumento de la tasa de muestreo del sistema es proporcional al tráfico en la red. Por ejemplo, según se mostró en la subsección anterior, el tráfico del sistema es de unos 300 Kbps promedio cuando está funcionando a 40hz. Sin embargo, esta medición sube hasta casi 1Mbps cuando se trabaja a 100hz, lo cual es explicable por el inundado de publicaciones en los tópicos correspondientes. Ahora bien, hay que tener en cuenta que aquí se presenta una decisión de compromiso: muestrear mejor para mayor estabilidad del sistema controlado o ahorrar ancho de banda perdiendo precisión en el control de la respuesta del sistema. Establecer la decisión óptima en este particular se encuentra fuera de los límites propuestos para este trabajo.

6.2.4 Influencia de los retardos

A continuación se presentan los resultados de simular distintos escenarios de red, esto es distintas combinaciones de retardo y jitter. Para trabajar con prolijidad se buscaron escenarios que tuvieran sentido y que se respaldaran en algún estándar, norma, recomendación o trabajo previo. Así, los cuatro escenarios que se utilizaron son:

1. Red LAN cableada con UTP cat 5 con menos de 1ms de retardo medio.
2. Red modelada con lo especificado como Clase 0 por la ITU1541 5 (100ms - 50ms).
3. Red modelada con lo especificado como Clase 1 por la ITU1541 (400ms – 50ms).
4. Red modelada con 200ms-100ms 7 .

6.2.4.1 Escenario 1

Resulta evidente que en estas condiciones se establece un buen control. También se ve la presencia de un pico a la subida y un error de régimen permanente. Ambas cosas pueden ecualizarse con una correcta calibración de los parámetros, cosa que el alumno puede ir probando desde la interfaz web. Un análisis interesante sobre este particular es el tiempo de

subida (t_sub) y cómo se puede disminuir mediante una combinación apropiada de las constantes (particularmente agregando "kd").

En la figura 15 puede verse una disminución del tiempo de subida mediante subir la constante derivativa (kd).

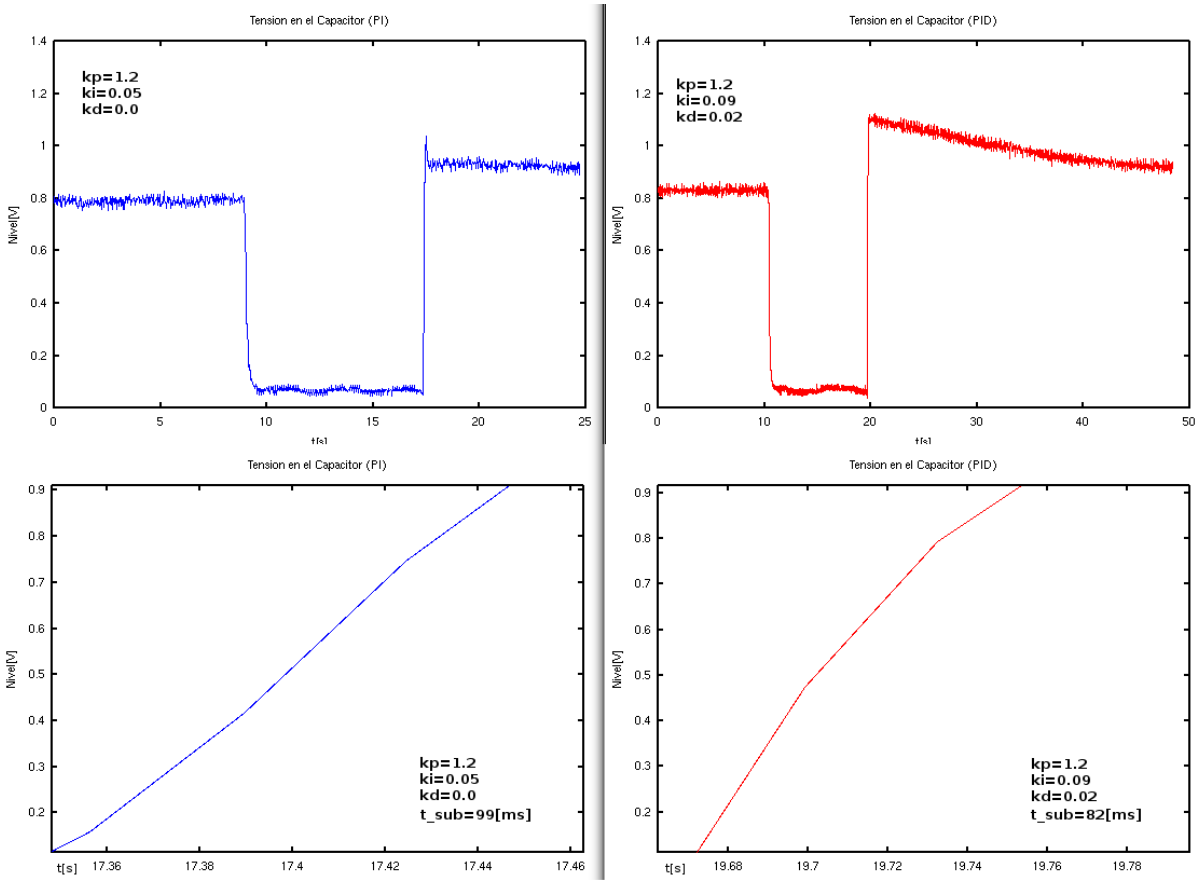

Figura 15: Tráfico en el sistema

Se reitera que dicha configuración no pretende ser la óptima, sino más bien un punto de análisis interesante y, en caso de ser operado por un alumno, un punto de partida para realizar un ajuste apropiado del PID.

6.2.4.2 Escenario 2

Puede hacerse otra prueba con valores correspondientes a una Clase 0 de la ITU. La figura 16 presenta la situación. Resulta evidente que, aunque hay cierto aumento "ruido" (debido a la dificultad para controlar debido al retardo) en comparación con el escenario 1, el sistema puede mantenerse estable (converge al set-point establecido) mediante el control del PID. Viendo el histograma se visualiza bastante "picudo" y centrado en el set-point, excepto por supuesto en los valores correspondientes a la perturbación (valores bajos ~0.1).

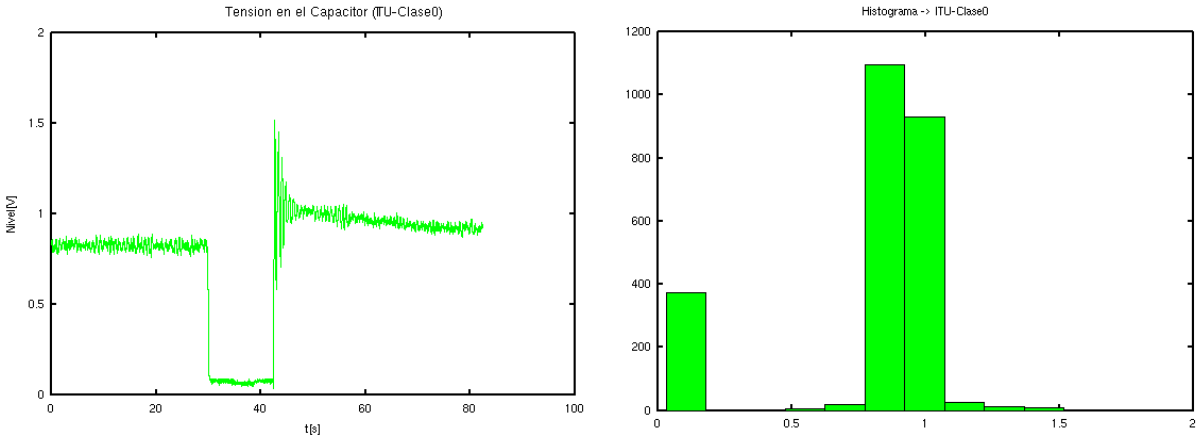

Figura 16: Respuesta del sistema (Nivel) a una perturbación e histograma (escenario 2)

6.2.4.3 Escenario 3

Los resultados de otra prueba, ahora con la configuración de red correspondiente a la Clase 1 de la ITU, se muestran plasmados en la figura 17. Puede apreciarse que el retardo es demasiado grande para poder realizar el control. En una red real, el retardo incluye la pérdida de paquetes que, gracias a TCP que genera sendas retransmisiones, se transforma en retardo. Puede observarse en el histograma que no se destaca ninguna tendencia media, lo cual se explica con la tendencia divergente del sistema en estas condiciones.

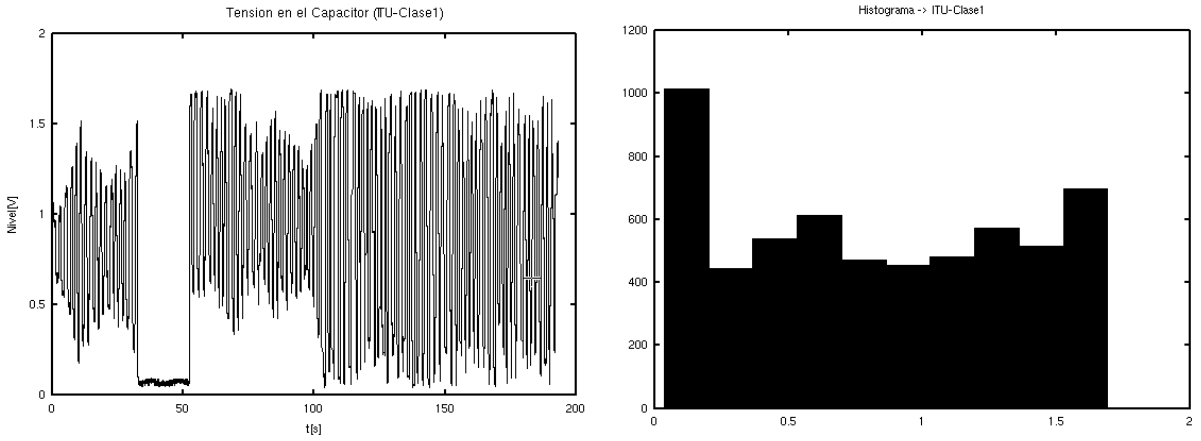

Figura 17: Respuesta del sistema (Nivel) a una perturbación e histograma (escenario 3)

6.2.4.4 Escenario 3

Se propuso en las pruebas un escenario intermedio, donde si bien se tiene un retardo intermedio entre las clases 0 y 1 de la ITU (delay≤200ms), se le aumenta al doble de jitter (jitter≤100ms). La figura 18 muestra los resultados de nivel de tensión e histograma correspondiente. Puede verse que si bien la dificultad para realizar el control es mayor que en el escenario 1, no llega a desestabilizarse el sistema. En el histograma se ve un consecuente "ensanchamiento", lo que corresponde a la mayor fluctuación de los valores de la salida.

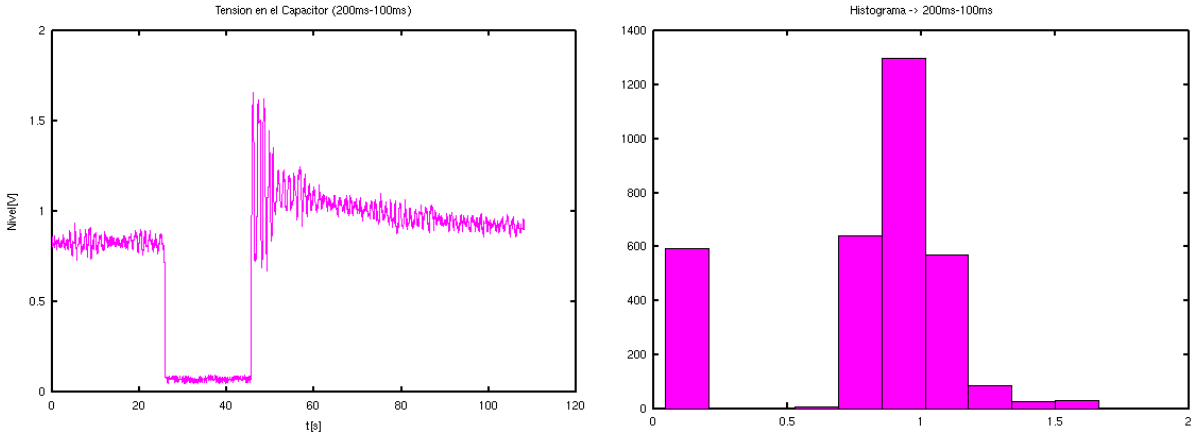

Figura 18: Respuesta del sistema (Nivel) a una perturbación e histograma (escenario 3)

7 CONCLUSIONES

En el presente trabajo se propuso una arquitectura de laboratorio remoto (LR) para enseñanza de las ciencias y la ingeniería. Para lograrlo se hizo un análisis desde lo general hasta lo específico que desembocó en la elección de un middleware de robótica que resulta apropiado para la realización de prácticas experimentales remotas.

Inicialmente se comentó que los laboratorios remotos son un tema de interés en la ingeniería y que por el diseño de los mismos no es descabellado pensarlos como una estructura de sistemas distribuidos y plantearlo dentro de una arquitectura de Middleware Orientado a Mensajes (MOM). Particularmente, se ha hecho la elección de ROS como un sistema plausible a poner en práctica en un caso de estudio de laboratorio remoto y, en el presente capitulo, se ha utilizado una microcomputadora de hardware embebido (Beaglebone) en relación con ROS.

Finalmente, se mostraron los resultados obtenidos al utilizar la arquitectura propuesta sobre un caso de estudio real. En estos se pudo visualizar la potencialidad del MOM ROS y sus capacidades en la implementación de nodos de procesamiento de datos e imágenes. Se presentó un modelo de laboratorio remoto completo que incluía sensado y actuación junto con un nodo de control (PID) y una interfaz gráfica de usuario basada en web; dejando sobre esta última constancia de las ventajas de utilizar HTML5-CSS3.

También se hizo un análisis de ancho de banda, quedando en evidencia que no hay requerimiento excesivo del mismo, con lo cual el alumnado, disponiendo de un navegador moderno y una conexión a internet decente, no tendría ningún problema para utilizar el LR. Lo que sí se estableció es que en rangos normales de Delay-Jitter (≤200ms - 100ms) y con una adecuada selección de tasas de muestreo, para sistemas con constantes de tiempo razonables (muestreables bien con Ts=40HZ o menos), ROS puede ser una plataforma viable para la implementación de un laboratorio remoto aplicado a la enseñanza de las ciencias y la ingeniería o también para aplicaciones industriales que no requieran tiempos de muestreo demasiado altos.

BIBLIOGRAFÍA

Callaghan MJ., J. Harkin J., Prasad G., MeGinnity TM., Maguire LP., Integrated Architecture

for Remote Experimentation, *IEEE International Conference on System Man and Cybernetics*, 2003

González, M.A., Adiego, J., Sanz, L.F., Bouab, N., Bouab,W., Mass,J., Laboratorios Remotos en la WEB, una Herramienta para la Cooperación al Desarrollo en el Campo de la Educación, *I Encuentro de Cooperación para el Desarrollo 2.0*, Gijón, España, 30 y 31 de Enero de 2008.

Liu, Y., Chen, C., Meng, M.,A Study on the Teleoperation of Robot Systems via WWW, Canadian Conference on Electrical and Computer Engineering, IEEE, 2000.

Guinaldo, M., Sanchez, J., Dormido, S., A Paket-based Network Control System Architecture for Teleoperation and Remote Laboratories, *49th IEEE Conference on Decision and Control*, USA, 2010.

Golberg, K., Mascha, M., Desktop Teleoperation via the World Wide Web," *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, May 1995.

Xue, X., Yang, S. X., Meng, M.Q.-H., Remote Sensing and Teleoperation of a Mobile Robot via the Internet, *Proceedings of the 2005 IEEE International Conference on Information Acquisition*, China, 2005.

Lum, M.J.H., Rosen, J., King, H., Friedman, D.C.W., Lendvay, T. S., Wright, A.S., Sinanan, M.N., Hannaford, B., Teleoperation in Surgical Robotics – Network Latency Effects on Surgical Performance, *31st Annual International Conference of the IEEE EMBS*, USA, 2009

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler,. R., Ng, A., ROS: an open-source Robot Operating System, *Proc. Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA), 2009*

Hirchie, S., Buss, M., Human-Oriented Control for Haptic Teleoperation, *Proceeedings of the IEEE*, vol 100, N°3, Págs. 623-647, 2012.

Jailly, B, Preda, M., Gravier, C., Fayolle, J., INTERACTIVE MULTIMEDIA FOR ENGINEERING TELE-OPERATION, *Multimedia and Expo (ICME), 2011 IEEE International Conference on*, Barcelona, 2011.

Curry, E., "Message-Oriented Middleware" in "Middleware for Communications." Edited by Mahmoud, Qusay, ISBN 0-470-86206-8, *Ed.: John Wiley & Sons* Ltd., 2004

McCafferty, B., Message-Based Systems for Maintainable, Asynchronous Development, *Blog on devlicio.us*, 2010.

Elkady, A., Sobh, T., Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography, *Journal of Robotics Volume 2012* (2012), Article ID 959013, 15 pages, 2012.

Wikipedia – http://www.wikipedia.com

Bishop, T., Karne, R. "A survy of Middleware", Computers and Their Applications, 2003

paginas.fe.up.pt (Consultado: 04/2014)

McCafferty, B., Message-Based Systems for Maintainable, Asynchronous Development. On *www.sharprobotica.com/page/4/.* Año 2010. (Consultado: 04/2014)

Tosco, S. J., Corteggiano, F., Broll, M., Implementación de un esquema de teleoperación utilizando el sistema operativo ros en el contexto de un laboratorio remoto, *Mecánica Computacional Vol XXXI, págs. 3741-3749*, Salta, Argentina, 13-16 Noviembre 2012

Tosco, S. J., Corteggiano, F., Broll, M., Teleoperación de un laboratorio remoto para la enseñanza a distancia de la ingeniería", *XV Reunión de Trabajo en Procesamiento de la Información y Control*, Río Negro, Argentina, 16 al 20 de septiembre de 2013.

Página de Remote-Lab: http://pr2-remotelab.com/ (Consultado: 04/2014)

Osentoski, S., Pitzer, B., Crick, C., Jay, G., Dong, S., Grollman, D., Suay, H. B., and Jenkins, O. C., Remote robotics laboratories for learning from demonstration: Enabling user interaction and shared experimentation", *International Journal of Social Robotics*, 2012.

Fayolle, J., Gravier C., Jailly B., Collaborative remote laboratory in virtual world, *In*

Proceedings of the 10th WSEAS on Applied Informatics and Communication, Taipei, Taiwan, August 20–22, 2010.
http://sourceforge.net/apps/mediawiki/mjpg-streamer/index.php?title=Main_Page (Consultado: 04/2014)
Página del Simposio Argentino de Sistemas Embebidos (SASE), *http://www.sase.com.ar/* (Consultado: 04/2014)
https://courses.cs.washington.edu/courses/cse466/12au/calendar/16f-ros_tutorial.pdf (Consultado: 04/2014)