Memorias del XVI Congreso Latinoamericano
de Control Automático, CLCA 2014
Octubre 14-17, 2014. Cancún, Quintana Roo, México

Laboratorio de Prácticas para la Enseñanza de Sistemas de Control de Tiempo Real

H. Vargas * **R. Costa-Castelló** ** **S. Pavez** * **G. Farias** *
G. Hermosilla * **J. Morales** * **S. Dormido** ***

* *Pontificia Universidad Católica de Valparaíso, Valparaíso, Chile.*
(e-mail: {hector.vargas, gonzalo.farias, gabriel.hermosilla,
jaime.morales}@ucv.cl, sebastian.pavez.t@gmail.com)
** *Universitat Politècnica de Catalunya, Barcelona, España.*
(e-mail: ramon.costa@upc.edu)
*** *Universidad Nacional de Educación a Distancia, Madrid, España.*
(e-mail: sdormido@dia.uned.es)

Resumen: El presente trabajo describe el diseño y desarrollo de un laboratorio de prácticas especialmente concebido para apoyar el aprendizaje en sistemas de control de tiempo real. La herramienta desarrollada permite realizar experiencias de control de tiempo real sobre un motor de corriente continua tanto en modo simulación (basado en el modelo del proceso) o bien realizando pruebas prácticas usando el motor físico. Así, mediante la asignación de *periodos de muestreo*, *tiempos de cómputo de tareas* y *prioridades de ejecución*, el usuario final de la aplicación (profesores y estudiantes) puede observar el comportamiento correcto o incorrecto del sistema de control permitiendo, por contraste, reafirmar los aspectos teóricos de la metodología de implementación de sistemas de tiempo real.

Keywords: Educación en Control, Sistemas de Tiempo Real, Laboratorios Virtuales y Remotos.

1. INTRODUCCIÓN

Una vez que el diseño de una estrategia de control ha sido correctamente probada mediante técnicas de simulación, esta debe ser implementada sobre el proceso para ser ejecutada en tiempo real. Desde el punto de vista de su implementación práctica, esta última etapa comúnmente involucra el correcto entendimiento y aplicación de conceptos de *sistemas de tiempo real*, tales como: *tarea*, *periodo de muestreo*, *concurrencia*, *prioridad*, *política de planificación*, *tiempo de cómputo*, *tiempo de activación*, *tiempo límite*, etc., los cuales no siempre son bien entendidos por estudiantes que se inician en el campo del control automático. Por ejemplo, ellos usualmente asumen que el controlador se ejecuta como un único proceso un computador dedicado exclusivamente a tal labor, lo cual en muchas situaciones no es cierto, ya que existen otros servicios o procesos que comparten los recursos computacionales del sistema.

Numerosas universidades alrededor del mundo ofrecen cursos de tiempo real en sus planes de estudio, generalmente en el ámbito de la informática (Litayem et al. (2011)). Sin embargo, al centrarse en el ámbito del control automático, su presencia es más escasa. En (Bradley et al. (2012)) se detalla una alternativa educacional que permite a los estudiantes participar en la concepción e implementación de un sistema de control en tiempo real. Por otra parte, también existen herramientas para la concepción de sistemas de tiempo real las cuales, dadas sus características, pueden ser adaptadas para su utilización en el ámbito docente (Bucher et al. (2009)).

Un aspecto en común de los trabajos referenciados previamente dice relación con su enfoque. En todos estos casos se aborda el diseño, desarrollo e implementación del sistema de tiempo real como eje central. El presente trabajo, sin embargo, posee motivaciones docentes desde una perspectiva diferente. Específicamente, se busca ofrecer al estudiante un ejemplo práctico que reafirme parte de la teoría de los sistemas de tiempo real. En la actualidad existen herramientas de simulación que cumplen este rol (Farias et al. (2010)). Sin embargo, en las simulaciones, a pesar de su potencial, la identificación con la experiencia por parte del alumno puede no llegar a ser tan poderosa o reafirmante. De esta forma, se busca elaborar una aplicación que junte tanto los aspectos de simulación como los de comportamiento práctico.

Basado en la discusión anterior, este artículo presenta el desarrollo de un laboratorio virtual y remoto especialmente concebido para el apoyo en la enseñanza de sistemas de control de tiempo real. La herramienta permite explorar la problemática de control que existe cuando se modifican algunos parámetros involucrados en la ejecución de un sistema de tiempo real y cómo estos afectan en la implementación práctica de un sistema de control.

Este trabajo se organiza de la siguiente manera. La sección 2 presenta un breve introducción a los sistemas de tiempo real. Luego, la sección 3 presenta las herramientas de hardware y software que se utilizaron para el desarrollo del laboratorio mientras que la sección 4 presenta su metodología de implementación. La sección 5 presenta las pruebas de funcionamiento realizadas al laboratorio y, finalmente, la sección 6 presenta las conclusiones finales.

2. CONCEPTOS DE TIEMPO REAL

Se considera a un sistema informático de tiempo real (RT) si su correcta funcionalidad depende no sólo de su lógica programada sino también del cumplimiento de unas especificaciones temporales (Burns and Wellings (2003)). Los sistemas de tiempo real pueden clasificarse en: *sistemas de tiempo real estricto*, en el cual la ejecución tardía (después de un plazo límite) puede provocar una falla crítica en todo el sistema, y *sistemas de tiempo real flexible* en los cuales algunos retrasos pueden ser tolerados, pero con algún grado menor de efectividad (por ejemplo, un mayor tiempo de estabilización en un sistema de control).

Normalmente los sistemas de tiempo real ejecutan concurrentemente diferentes tareas, incluyendo tareas de control. Así la selección de qué tarea que debe ser ejecutada depende de la política de planificación utilizada. El planificador es un algoritmo que decide qué tarea debe ejecutarse en cada instante. Para ello, este algoritmo requiere conocer las características de cada tarea que compone el sistema. Las características más comunes son las siguientes:

- *Tiempo de activación (o liberación)*: Momento en el que, idealmente, la tarea debería iniciar su ejecución.
- *Tiempo límite* : Momento en el que la tarea debe haber finalizado su ejecución.
- *Tiempo de cómputo (o ejecución)*: Tiempo que tarda el computador en ejecutar el código de la tarea.
- *Periodo*: Tiempo entre dos instantes de activación consecutivos.
- *Plazo*: Tiempo entre el instante de activación y el instante límite de finalización. Comúnmente el plazo corresponde al periodo de una tarea periódica.
- *Prioridad*: Preferencia de ejecución de una tarea respecto al resto.

Las tareas pueden estar en uno de los tres siguientes estados: ejecutando (**R**, la tarea quiere ejecutarse y tiene la CPU asignada), en espera (**P**, la tarea quiere ejecutarse y pero no tiene la CPU asignada) y (**S**, bloqueado a la espera de un evento temporal o externo). El primer estado indica que la tarea se está ejecutando en la CPU. Si hay una tarea que debería ejecutarse, pero que no puede porque existe otra con mayor prioridad utilizando la CPU se dice que la tarea está apunto. Si una tarea está esperando que se libere un recurso se dice está bloqueda. Finalmente si una tarea decide paralizar su ejecución a la espera que pase un cierto tiempo se dice que está durmiendo. La Figura 1 muestra los estados de tres tareas.

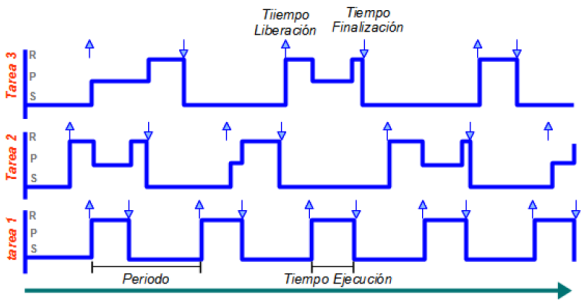


Figura 1. Monitor de planificación: 3 tareas ejecutadas en paralelo en el mismo procesador.

Las políticas de planificación en tiempo real suelen agruparse en políticas de prioridades estáticas o de prioridades dinámicas. RMPA y DMPA son dos maneras de asignar prioridades utilizadas en planificación estática(Burns and Wellings, 2003). La primera asigna mayor prioridad a aquellas tareas con un periodo menor mientras que la segunda asigna mayor prioridad a aquellas tareas con un plazo menor; en ambos casos la asignación se realiza fuera de línea. Earliest Deadline First (EDF) es un política conocida de planificación dinámica(Burns and Wellings, 2003). Esta da mayour prioridad a aquellas tareas con un timepo límite más cercano, este cálculo se realiza en línea.

En las secciones siguientes se describe el diseño e implementación del laboratorio virtual y remoto desarrollado para el estudio de los conceptos de tiempo real explicados en este apartado.

3. HERRAMIENTAS DE HARDWARE Y SOFTWARE PARA EL DESARROLLO DEL LABORATORIO

En esta sección se describe el dispositivo hardware utilizado como prototipo de pruebas para el desarrollo del laboratorio, en este caso, un *motor de corriente continua* especialmente concebido para la práctica docente del control automático. Seguidamente, se presentan las herramientas de software necesarias para la implementación del control en tiempo real tanto desde la perspectiva de la simulación como del sistema físico en el laboratorio.

3.1 Motor de corriente continua

Una de las máquinas eléctricas más comunes y estudiadas son los motores de corriente continua. Sus cualidades de alto torque de arranque, su relativa alta velocidad de respuesta, así como la facilidad para controlarlo por medios eléctricos y electrónicos lo convierten en un objetivo ideal para el estudio del control. La Figura 2 muestra la disposición real del dispositivo en el laboratorio.

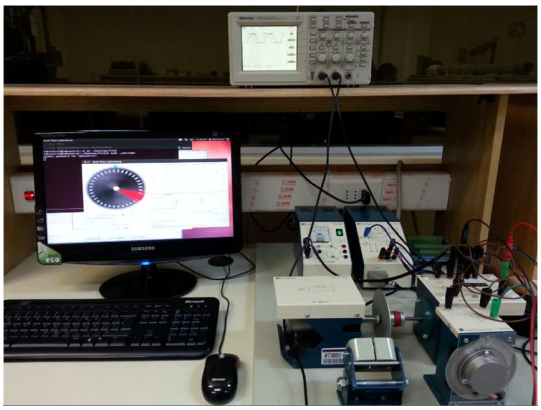


Figura 2. Motor DC MS-150 de FEEDBACK.

El proyecto utiliza el Servosistema Modular MS-150 de FEEDBACK, empresa especializada en la fabricación de equipamiento didáctico para la enseñanza del control automático. De los diferentes módulos disponibles, se utiliza el motor DC (módulo DCM150F), el tacómetro (módulo GT150X), el lector de posición (módulo OP150K) y el módulo que maneja el sentido de giro del motor (módulo SA150D), el cual se realiza con un control por armadura.

Para el manejo de datos desde y hacia el motor, se utiliza la tarjeta de adquisición de datos PCI-1711U de Advantech, que es instalada en un computador personal desde donde se controla el dispositivo de forma local (ver Figura 4).

Para la implementación del control del motor (que será tratada en la sección 4.2) es necesario contar con un modelo del proceso que permita diseñar los controladores tanto de velocidad como de posición. A partir de la aplicación de un escalón unitario a la entrada del motor es posible determinar una función de transferencia de primer orden que modele la planta, considerando motor, armadura y tacómetro como un bloque (ver Figura 3).

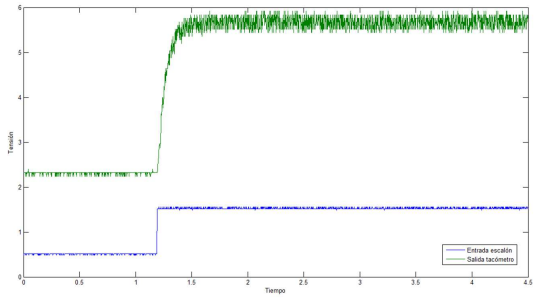


Figura 3. Escalón unitario aplicado al motor.

La identificación del modelo se realizó para el punto de operación $U = 0,5\ [voltios]$ y $V = 2,25\ [voltios]$. De este proceso se ha obtenido el siguiente modelo:

$$\frac{V(s)}{U(s)} = \frac{3,733}{0,07483s + 1}. \quad (1)$$

Cabe mencionar que la función de transferencia para el control de posición del motor se obtiene incorporando un integrador a la ecuación 1.

3.2 Herramientas de software utilizadas

Para desarrollar un sistema de tiempo real se hace necesario disponer de ciertas condiciones especiales tales como reducción de latencias, concurrencia de tareas o un adecuado manejo de las prioridades de ejecución. De esta forma, y dado que un sistema operativo tradicional generalmente no se ajusta a estas necesidades, se hace uso de un sistema operativo de tiempo real. En este sentido, el desarrollo de tiempo real que se presenta en este trabajo utiliza XENOMAI (Xen (2014)), un sistema operativo de tiempo real de código libre basado en Linux el cual ha sido ampliamente adoptado por la comunidad internacional de desarrolladores de aplicaciones de tiempo real.

Xenomai, en base a un mismo Kernel, permite utilizar diferentes API de programación. Entre ellas destacan POSIX, VxWorks, VRTX, RTAI, entre otras. Además, ofrece facilidades para migrar códigos entre ellas. La interfaz a utilizar en este trabajo es POSIX (Portable Operating System Interface), la cual corresponde a un estándar IEEE que busca establecer una compatibilidad entre los sistemas operativos UNIX. En este estándar se pueden distinguir diferentes grupos de trabajo, donde el relacionado con los aspectos de sistemas en tiempo real se denomina POSIX-4 (Gallmeister (1995)). Éste se centra principalmente en la gestión de los hilos de ejecución (threads o tareas), la comunicación entre ellos, mecanismo de planificación de ejecución temporal, entre otros servicios. POSIX se programa escencialmente en código C.

Por otra parte, el trabajo también considera la implementación de un entorno simulado de control de tiempo real. Para ello se ha escogido *Easy Java Simulations* (EJS), una herramienta de software gratuita diseñada para la creación de simulaciones interactivas en lenguaje Java (Esquembre (2014)). El usuario al que está dirigida EJS son estudiantes, profesores e investigadores de ciencias e ingeniería, que poseen un conocimiento básico de programación de computadores, pero que no disponen de una gran cantidad de tiempo para crear una simulación gráfica con un elevado grado de interactividad.

En conjunto con EJS, este desarrollo utiliza la librería JTT (Java True Time) que proporciona una API especialmente concebida para la creación de entornos de simulación de sistemas de tiempo real (Farias et al. (2010)).

4. DETALLES DE IMPLEMENTACIÓN

Dado que se pretende crear un módulo de prácticas que reúna el entrenamiento tanto en modo simulado como sobre el dispositivo físico real, se ha planteado la creación de un laboratorio remoto como módulo de trabajo complementario. Las siguientes secciones describen cada una de las etapas de implementación del laboratorio.

4.1 Arquitectura del laboratorio virtual y remoto

La Figura 4 presenta la arquitectura de comunicación del laboratorio y su relación con las herramientas de software utilizadas para su desarrollo.

En relación a la implementación del lado del servidor, el intercambio de datos entre la planta y el computador se lleva a cabo mediante una tarjeta de adquisición de datos Advantage PCI-1711, se cierra el lazo de control y se configura el socket servidor que queda a la espera de conexión por parte del cliente.

La implementación del lado del cliente diferencia tres módulos desde el punto de vista de su programación: La construcción de la GUI o interfaz con la cual los usuarios llevan a cabo sus tareas de experimentación, la definición del modelo necesario para desarrollar una representación

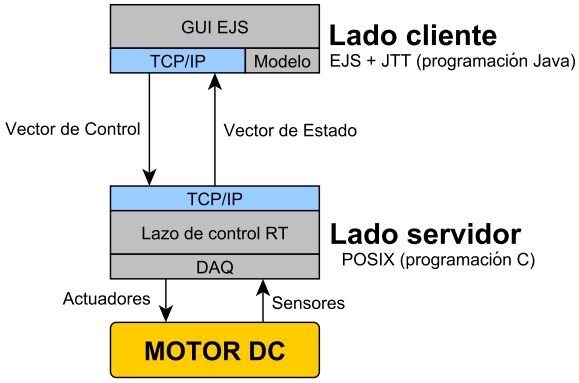


Figura 4. Arquitectura de comunicación cliente-servidor.

virtual del proceso y finalmente, la programación de los métodos de comunicación (sockets TCP/IP) necesarios para enlazar con la aplicación del lado del servidor.

4.2 Control en tiempo real mediante POSIX

Se diseñan controladores PID (Åström and Hägglund (2006)) para regular tanto en velocidad como en posición el eje del motor DC utilizado como dispositivo de pruebas. Se busca disponer de controladores simples ya que el énfasis en este trabajo radica en su desempeño en tiempo real. Se utiliza para el control de velocidad un PI y para el control de posición un PD, aprovechando el integrador existente en el modelo de la planta.

El diseño del controlador de velocidad se realiza de forma analítica mediante el método de asignación de polos. Se busca disponer de un tiempo de asentamiento de 0,5 [seg] sin sobrepaso. Esto se logra mediante las ecuaciones que relacionan el sobrepaso (OS) y el tiempo de asentamiento (T_a al 2 %) con la razón de amortiguamiento (ζ) y la frecuencia natural (w_n) (ver ecuaciones (2) y (3)).

$$\%OS = 100 \cdot e^{\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}} \tag{2}$$

$$T_a = \frac{4}{\zeta w_n} \tag{3}$$

Así, para un tiempo de asentamiento de 0,5 [seg] y un sobrepaso nulo se tienen valores de $\zeta = 0,98$ y $w_n = 4,7$. Con estos valores es posible utilizar las fórmulas que relacionan la ecuación característica de lazo cerrado ($s^2 + 2\zeta w_n s + {w_n}^2$) con la de la planta junto al controlador PI, las cuales se aprecian en las ecuaciones (4) y (5).

$$K_p = \frac{2\zeta w_n \tau - 1}{K} \tag{4}$$

$$T_i = \frac{2\zeta w_n \tau - 1}{{w_n}^2 \tau} \tag{5}$$

De esta manera, se obtienen tanto la constante proporcional $K_p = 0,0737$ $[voltios]$ como el tiempo integral $T_i = 0,0487$ $[seg]$ para el controlador PI.

Para el caso del controlador PD de posición se utiliza el método basado en el criterio ITAE, aprovechando las fórmulas que relacionan la función característica en lazo cerrado con la frecuencia natural (Martins (2005)). De esta forma, si se desea un sobrepaso menor al 5 % con un tiempo de asentamiento bajo el medio segundo, se pueden obtener los parámetros proporcional y derivativo del controlador $K_p = 2,88$ $[voltios]$ y $T_d = 0,024$ $[seg]$.

Seguidamente, ambos controladores fueron programados en POSIX (código C) con el fin de realizar pruebas de desempeño en tiempo real de los controladores diseñados. Para ello, se programa un *thread en tiempo real* cuyo diagrama de flujo se presenta en la Figura 5.

El código de control a ejecutarse de forma periódica se dispone dentro de un bucle *while()*. Para ello, se fija un periodo de ejecución al inicio del código de la tarea, el que luego es referenciado por una instrucción de espera al final del bucle. El tiempo de cómputo del código dentro de dicho bucle, por tanto, no debe ser mayor al período definido, ya

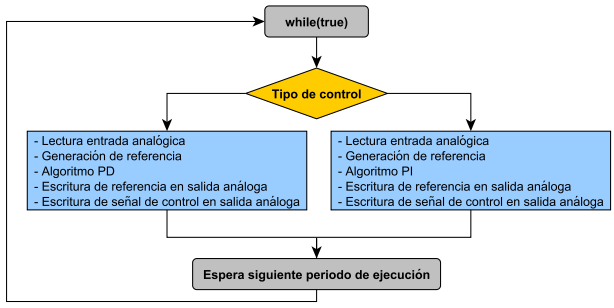


Figura 5. Acción de control a ejecutarse periódicamente.

que la instrucción de espera haría referencia a un momento de tiempo pasado. Lo anterior se ve reflejado en que, conforme el tiempo de cómputo de la tarea se aproxima a su período asignado, el porcentaje de utilización de CPU se incrementa hacia el 100 % tal y como se aprecia en la Figura 6. Xenomai implementa un sistema *watchdog* que detiene la ejecución del programa ante condiciones no adecuadas.

```
labcontrol@labcontrol:~$ cat /proc/xenomai/stat
CPU  PID    MSW        CSW        PF    STAT       %CPU  NAME
  0  0      0          1967936    0     00500088    6.2  ROOT/0
  1  0      0          0          0     00500080  100.0  ROOT/1
  2  0      0          0          0     00500080  100.0  ROOT/2
  3  0      0          0          0     00500080  100.0  ROOT/3
  4  0      0          0          0     00500080  100.0  ROOT/4
  5  0      0          0          0     00500080  100.0  ROOT/5
  6  0      0          0          0     00500080  100.0  ROOT/6
  7  0      0          0          0     00500080  100.0  ROOT/7
  0  4839   3          3          0     00b00380    0.0  control -> main()
  0  4841   1          33920      0     00300180   93.7  control -> pert1()
  0  4842   32959      38186      0     00b00380    0.0  control -> control()
  0  0      0          4142467    0     00000000    0.0  IRQ2312: [timer]
```

Figura 6. Porcentaje de utilización de CPU.

Cabe mencionar que el periodo de muestreo asignado a la tarea de control (cuyo algoritmo básico de ejecución se presenta en el listado 1) ha sido de 2 [ms], lo cual busca hacer el funcionamiento del sistema más exigente. Las razones de ello se relacionan con las pruebas de funcionamiento que se presentan en la sección 5 de este trabajo, las que ilustrarán el desempeño de los controladores desde una perspectiva de tiempo real.

Listing 1. Algoritmo PID.

```
N = 100;
h = PERIODO*0.000000001;
P = kp*(Vref-Vi);
I = I0;
D = td/(N*h+td)*D0 + N*kp*td/(N*h+td)*(Vi0-Vi);
ctr = P + I + D;
I0 = I0 + kp*h/ti*(Vref-Vi);
D0 = D;
Vi0 = Vi;
```

Dado que el fin último de este trabajo es ilustrar, de forma práctica, el efecto de una correcta parametrización de las tareas de un sistema de control de tiempo real, es que se propone, como mecanismo de prueba, la incorporación de una tarea perturbadora que corra en paralelo a la tarea de control. De esta manera, mediante la asignación de *periodos de muestreo*, *tiempos de cómputo* y *prioridades* a las tareas de control y perturbación, se podrá observar el comportamiento correcto o incorrecto del sistema de control permitiendo, por contraste, reafirmar los aspectos teóricos de la metodología de tiempo real.

Así, se busca que la tarea perturbadora posea, además de una ejecución periódica, un tiempo de cómputo variable. Esto último se regula mediante el número de iteraciones de un bucle *for()*. Por otro lado, dado que el tiempo utilizado por la tarea de control es muy bajo, el tiempo de cómputo de la tarea de perturbación debe alcanzar valores muy cercanos a su periodo para afectarlo. Con tal de evitar la detención de la ejecución por parte del *watchdog* de Linux, se aumentó el tiempo de cómputo de la tarea de control a 0,15 [ms], también por medio del número de iteraciones en un ciclo *for()*. Naturalmente, esto se realiza sin degradar el desempeño de los controladores.

La idea principal es asignar una mayor prioridad al thread de perturbación que al de control y aumentar su tiempo de cómputo para visualizar el efecto que produce en esta última. Después de ello, un cambio en las prioridades asignadas, buscará poner de manifiesto los beneficios de una mayor prioridad a una tarea crítica, en este caso la tarea de control de un motor. Se debe mencionar que las prioridades utilizadas son fijas, dado que Xenomai sólo permite este tipo de asignación.

De esta forma, se dispone de un periodo de ejecución para ambas tareas de 2 [ms] (ver Figura 7). Así se considera que, de forma teórica, a partir de un tiempo de cómputo de 1,85 [ms] para la tarea perturbadora, se aprecia una influencia en la tarea de control. Además de este caso, se presentan las situaciones con 1,9 [ms] y 1,95 [ms] con una prioridad mayor para el thread perturbador. Finalmente, se asigna la tarea de control al dominio de Xenomai manteniendo un alto tiempo de cómputo en la tarea perturbadora para ver sus efectos en el control.

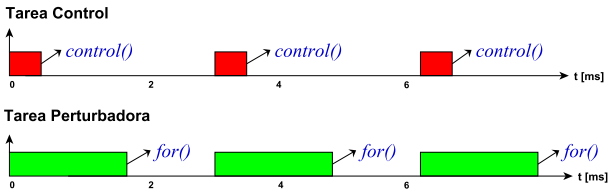


Figura 7. Representación gráfica de las tareas en el tiempo.

4.3 Simulación del control de tiempo real en EJS

Como ya se ha mencionado, el laboratorio a implementar consta de dos partes: la simulación del sistema y la conexión remota. En esta sección se discute el primer rasgo, con el que se busca reforzar el conocimiento que esta herramienta pueda brindar al estudiante, de manera que sea posible realizar un análisis del sistema desarrollado en base a simuladores de tiempo real. En particular, y como se mencionó en la sección 3.2, este trabajo utilizó la herramienta de simulación EJS y la librería JTT para desarrollar el entorno simulado del sistema de control de tiempo real para el motor DC.

Un programador puede utilizar la librería JTT directamente en un programa Java, o utilizando EJS para crear simulaciones de sistemas de tiempo real. La integración de la librería con otras herramientas se observa en la Figura 8. El entorno de simulación del sistema de tiempo real está dado por JTT, mientras que el solver y la vista (interfaz de usuario) son suministrados por otras herramientas

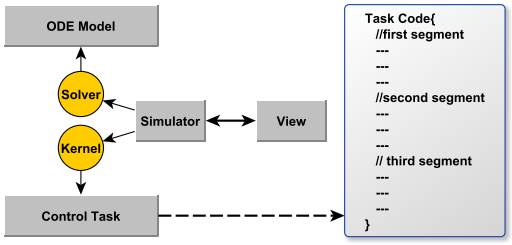


Figura 8. Simulación en tiempo real utilizando JTT.

Java, en este caso EJS. La sección 4.5 presentará la interfaz de usuario final del entorno de simulación desarrollado.

Para conocer más detalles del uso de la librería JTT y EJS, el lector puede revisar la referencia (Farias et al. (2010)).

4.4 Comunicación cliente-servidor

La Figura 9 muestra un diagrama representativo de la lógica de comunicación cliente-servidor del laboratorio. Al programa constituido por la tarea de control y perturbación se le incorpora un nuevo *thread*. Esta nueva tarea, con un periodo de 40 [ms] y ejecución en el kernel de Linux, cumple un rol de servidor TCP/IP tradicional. Con ella se busca que un usuario pueda visualizar las variables del sistema y además comandar la ejecución del mismo. Así, las dos tareas mencionadas pasan a estar administradas por este thread servidor. Según el valor que es recibido desde el cliente, la acción a ejecutar será diferente por parte del thread servidor. Se tienen cuatro casos:

- Crear las tareas de perturbación y de control con una mayor prioridad para esta última.
- Crear las mismas tareas pero con una prioridad mayor para la perturbadora.
- Destruir el thread perturbador y de control. Se ejecuta una tarea encargada de apagar civilizadamente el motor. Esta acción es necesaria para crear las tareas nuevamente.
- Finalizar la conexión con el cliente, quedando el servidor disponible para establecer una nueva conexión. Se destruyen las tareas existentes.

Aquí se hace necesario también compartir datos entre las tareas. Para esto se utilizan variables globales y semáforos. Se tiene, así, un intercambio de datos entre: la tarea de control y el servidor y entre esta última y la tarea de perturbación. Esta información es enviada hacia el

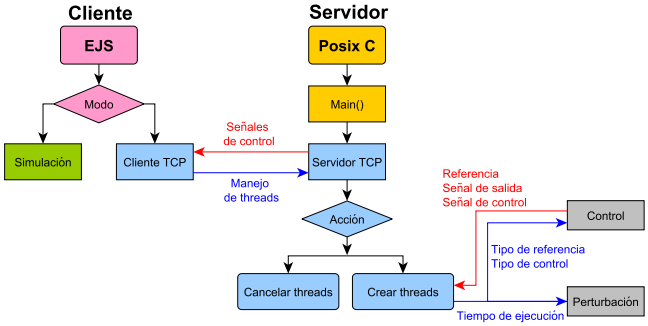


Figura 9. Proceso de conexión entre servidor y cliente.

cliente EJS (referencia, señal de control y de salida) o bien recibida desde el mismo (iteraciones, tipo de referencia y tipo de control). Por su parte, el cliente EJS genera o utiliza estos datos, por medio de una interfaz de usuario, que además incluye la simulación. En ella se pueden graficar las variables del sistema, crear o destruir las tareas de control y perturbación, cambiar sus prioridades y modificar el tiempo de cómputo de la perturbación.

4.5 Aspecto final del laboratorio desarrollado

En la Figura 10 se muestra la interfaz de usuario lograda funcionando en conexión remota. Se disponen dentro de ella, tanto para la simulación como para el funcionamiento práctico, diferentes atributos, los que apuntan a facilitar la interacción con el estudiante. Por una parte, se tiene la representación del giro del motor. Además, se presentan las gráficas de las variables del sistema, la posibilidad de seleccionar el tipo de control y el tipo de referencia. En esta misma línea, se exponen los parámetros usados en los controladores. No se ofrece, sin embargo, la posibilidad de modificar estos últimos ya que se escapa de la intención de la aplicación. De todas formas, se entiende que este y otros atributos puedan ser añadidos en el futuro.

Respecto a los rasgos de tiempo real en sí, se ofrecen dos paneles. En uno de ellos, es posible visualizar una gráfica de planificación. Este elemento se considera un recurso clarificador para los estudiantes con el fin de entender el efecto del cambio del tiempo de cómputo y prioridades.

Finalmente, se dedica una porción de la interfaz para modificar los atributos temporales y de prioridad de las tareas de control y perturbación, lo que permite al usuario final experimentar probando los efectos de esta parametrización en el desempeño del control del motor.

5. PRUEBAS DE FUNCIONAMIENTO

La aplicación desarrollada busca complementar el estudio teórico de los sistemas de control de tiempo real. En primer lugar, se pretende que el estudiante interactúe con la simulación para formarse una idea del comportamiento ideal del sistema y luego, con la ejecución práctica, contrastarlo. Todo esto siguiendo como base las pruebas de funcionamiento descritas a continuación.

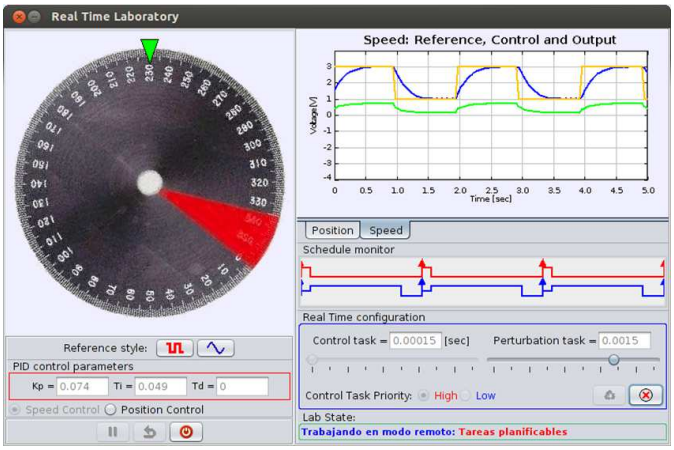
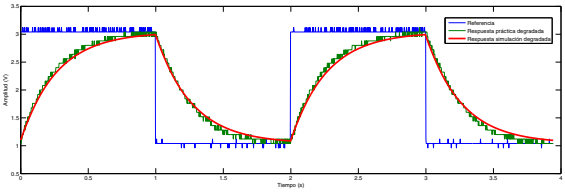


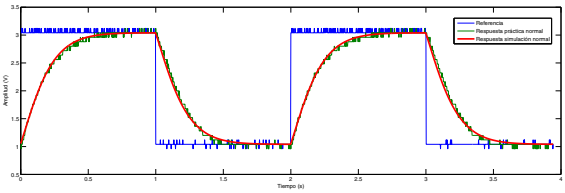
Figura 10. Interfaz gráfica del laboratorio de tiempo real.

5.1 Control de velocidad: Simulación v/s Práctica

Las Figuras 11 y 12 muestran la respuesta del controlador de velocidad y la correspondiente señal de control a un cambio en la referencia. Estas gráficas ilustran el contraste entre el comportamiento simulado del sistema y su homólogo práctico. Las imágenes superiores representan el funcionamiento con una prioridad mayor para la perturbación y, las inferiores, el desempeño al cambiar las prioridades. Ambos casos con un tiempo de cómputo para la tarea perturbadora de 1,95 [ms]. En las gráficas es posible apreciar la degradación del funcionamiento del sistema y su posterior mejora.

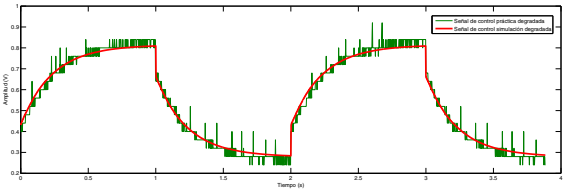


(a) Prioridad mayor para la tarea perturbadora.

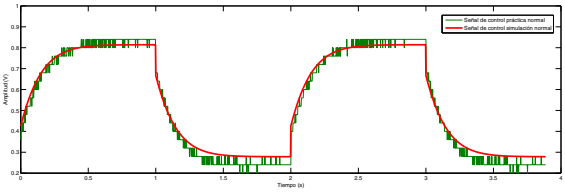


(b) Prioridad mayor para la tarea de control.

Figura 11. Evolución de la velocidad (simulación y experimental) y la referencia de velocidad.



(a) Prioridad mayor para la tarea perturbadora.

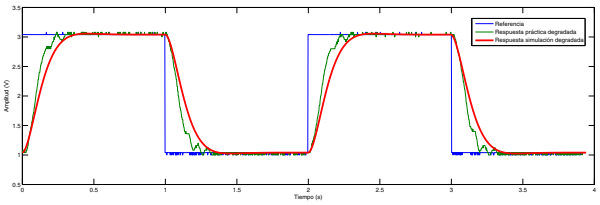


(b) Prioridad mayor para la tarea de control.

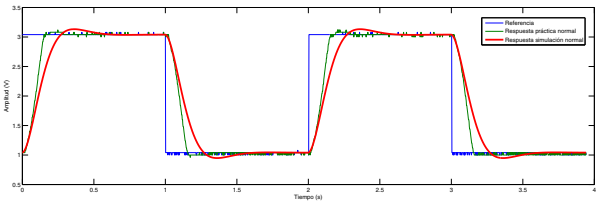
Figura 12. Evolución de la acción de control (simulación y experimental).

5.2 Control de posición: Simulación v/s Práctica

Al igual que en la sección anterior, se presenta en las Figuras 13 y 14 los resultados para el control de posición. Además de las diferencias de comportamiento ya señaladas, se puede observar que la degradación en la simulación se relaciona con una respuesta más lenta del sistema. En la acción de control, por otra parte, se aprecia un leve "rizado" en la porción transitoria de la señal al tener la tarea perturbadora la mayor prioridad.

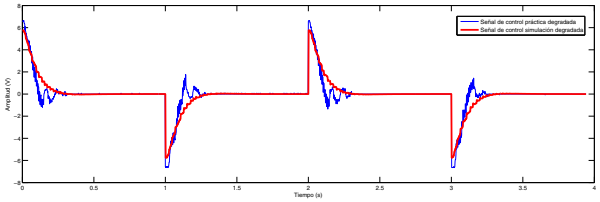


(a) Prioridad mayor para la tarea perturbadora.

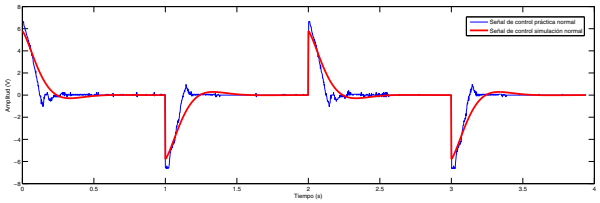


(b) Prioridad mayor para la tarea de control.

Figura 13. Seguimiento a referencia en posición.



(a) Prioridad mayor para la tarea perturbadora.



(b) Prioridad mayor para la tarea de control.

Figura 14. Evolución de la acción de control en el control de posición.

5.3 Análisis final

Conforme a los resultados obtenidos es posible crear una tabla resumen de las pruebas desarrolladas (ver Cuadro 1). Se debe recordar que para ambas tareas se tiene un periodo de 2 [ms]. El tiempo de cómputo de la tarea de control es de 0,15 [ms] y el de la tarea perturbadora es variable por medio del número de iteraciones de un ciclo *for()*. Con tal de que el tiempo asignado no genere inconvenientes en la ejecución del programa, se limita el máximo tiempo de cómputo asignado a la tarea perturbadora a 1,95 [ms].

Como se ha mencionado anteriormente, conforme el tiempo de cómputo de la tarea perturbadora se incrementa, la tarea de control comienza a verse afectada en su desempeño. Esta situación se revierte por medio de un cambio en la asignación de prioridades. De esta forma se tiene un funcionamiento normal de los controladores aún en presencia de otra tarea demandante de recursos. Por otra parte, se detalla el consumo de CPU relativo a cada tarea cuando se encuentra en el dominio de Xenomai. Este porcentaje se relaciona, como se discutió en la sección 4.2, con la relación entre el tiempo de cómputo de la tarea y el periodo asignado. De esta forma se tiene una idea de cuantos recursos quedan disponibles para las demás tareas presentes en el dominio de Linux y que son ejecutadas en base a una política equitativa.

Cuadro 1. Resumen de las pruebas

Tarea de mayor prioridad	Tiempo de cómputo tarea perturbadora	Consumo CPU	Control
Perturbadora	1,85 [ms]	91,1 %	Afectado
Perturbadora	1,90 [ms]	93,7 %	Afectado
Perturbadora	1.95 [ms]	96,7 %	Afectado
Control	1,95 [ms]	6,2 %	Normal

6. CONCLUSIÓN

Se presentó una aplicación dedicada a reforzar, de forma práctica, ciertos aspectos teóricos de los sistemas de tiempo real. Ésta se enfoca, principalmente, en destacar la influencia de la asignación de prioridades en tareas críticas de funcionamiento que involucran el control de sistemas.

Además, mediante su extensión a un laboratorio remoto, se obtuvo como producto una herramienta docente poderosa para la enseñanza de sistemas de control de tiempo real. Así mismo, la herramienta ofrece múltiples alternativas de desarrollo a futuro, como por ejemplo, su aplicación al manejo de plantas más exigentes en cuanto a su comportamiento temporal y funcionalidades críticas. En la misma línea, la disposición de una interfaz de usuario en conexión con Xenomai brinda la posibilidad de aplicar este esquema de monitoreo a otros proyectos futuros.

REFERENCIAS

(2014). Xenomai homepage available at. URL `http://www.xenomai.org/`.

Åström, K. and Hägglund, T. (2006). *Advanced Pid Control*. ISA-The Instrumentation, Systems, and Automation Society.

Bradley, P., Puente, J., Zamorano, J., and Brosnan, D. (2012). A platform for real-time control education with lego mindstorms. In *9th IFAC Symposium Advances in Control Education*. Nizhny Novgorod, Russia.

Bucher, R., Mannori, S., and Netter, T. (2009). Rtai-lab tutorial: Scilab, comedi, and real-time control, https://www.rtai.org/rtailab/rtai-lab-tutorial.pdf.

Burns, A. and Wellings, A. (2003). *SISTEMAS DE TIEMPO REAL Y LENGUAJES DE PROGRAMACION*. ADDISON-WESLEY, 3^{era} edición edition. ISBN : 9788478290581.

Esquembre, F. (2014). Easy java simulations homepage available at. URL `http://www.um.es/fem/EjsWiki/`.

Farias, G., Cervin, A., Arzen, K., Dormido, S., and Esquembre, F. (2010). Java Simulations of Embedded Control Systems. *Sensors*, 10(9), 8585–8603.

Gallmeister, B.O. (1995). *POSIX.4: programming for the real world*. O'Reilly and Associates, Inc. Sebastopol, USA.

Litayem, N., Achballah, A., and Saoud, S. (2011). Building XenoBuntu Linux Distribution for Teaching and Prototyping Real-Time Operating Systems. *International Journal of Advanced Computer Science and Applications*, 2.

Martins, F. (2005). Tuning pid controllers using the itae criterion. *International Journal of Engineering Education*, 21(5), 867–873.