Tesis de Magíster en Ingeniería

Arquitecturas Flexibles para Sistemas de Control de Tiempo Real

Martín Leandro Duval

2012

Departamento de Ingeniería Eléctrica y de Computadoras

Universidad Nacional del Sur

Agradecimientos

Aun no puedo creer que hayamos podido lograr este gran desafío que yo mismo me he propuesto. Digo hayamos, porque estoy convencido que no hubiera sido posible, sin la gran ayuda que he recibido de mis tutores y compañeros de laboratorio y sin el apoyo y amor brindado por mi familia y amigos.

En primer lugar, quiero agradecer al Dr. Ricardo Cayssials, *Richi* mi gran amigo, no solo por ser mi guía en todos los trabajos desarrollados que posteriormente dieron forma a esta tesis, si no también por las largas charlas de vida que tenemos y por aguantar todos mis miedos y complicaciones.

Al Dr. Javier Orozco quien allá por el año 2003 confió en mí para enviar el primer pedido de beca en momentos donde no encontraba el rumbo en mi profesión y todas las puertas se me cerraban. En ese entonces, no me la otorgaron, la desilusión fue grande y fue él quien me alentó. Recién en el tercer y último intento, ya con 29 años cuando el límite de edad para la presentación era de 30 años, me otorgaron la beca para poder concretar este sueño que hoy es realidad.

A todos mis compañeros del Laboratorio de Sistemas Digitales, Rodrigo Santos, Leonardo Ordinez, David Donari, José Urriza, Enrique di Pietro, Fernando

Otero, Omar Alimenti, Edgardo Ferro, quiero agradecerles por todos los buenos momentos y experiencias compartidas.

Especialmente quiero agradecerle a mi gran amiga Diana Sanchez por su amistad y su comprensión pero principalmente por sus principios y valores los cuales comparto firmemente.

A mis viejos, por el esfuerzo que han hecho toda su vida por domesticar a tres salvajes. Por inculcarnos desde pequeños esos grandes valores que hicieron que hoy seamos hombres de bien.

A mis suegros, por haber encontrado en ellos el cariño de una nueva familia, pero principalmente por haber educado con tanto amor a esas dos grandes mujeres y madrazas.

A mi tía Elena y mis primas Belén, Cele y Jose quiero agradecerles por todo el amor y la ayuda que nos dan a diario en la crianza de nuestros niños.

A mi padrino, por haberme ayudado tanto en mi etapa de estudiante de grado, *gracias tío!!!* sin tu ayuda todo hubiese sido muy complicado.

A mis hermanos Fabio y Mariano quiero decirles que les agradezco por haberme dejado cumplir tan bien esa hermosa tarea de ser el hermano mayor y por haber compartido tantos hermosos momentos.

A mi cuñado Nano por haber encontrado en el no solo un cuñado si no otro hermano y amigo.

A mis cuñadas Carina, Vanesa y Soledad, a mis sobrinos, Magali, Julieta, Olivia, Nicole, Benjamín y Bautista que son una parte muy importante en mi vida.

Por último, pero en el lugar más importante de mi corazón, quiero agradecer a mi hermosa familia. A mi compañera de vida, mi amiga, mi hermana, mi mujer, mi amor Yanina, con quien comparto el enorme desafío diario de ser padres. Hace unos días recordaba que cuando comencé a escribir esta tesis mi primer hijo Santino era bebe, luego llegó Catalina y más tarde Agostina. Con esto remarco lo dificultoso que fue este camino. A vos *Yani* gracias por esas hermosas criaturas que me diste y por tu aguante. *Santi, Cati y Agos,* gracias por darme tanto *AMOR* y por dejarme ser su papá.

A todos ustedes les digo simplemente *GRACIAS!!!*.

Prefacio

La presente Tesis fue presentada como parte de los requisitos para optar al grado Académico de *Magíster en Ingeniería* de la Universidad Nacional del Sur y no ha sido presentada previamente para la obtención de otro título en esta Universidad u otras.

La misma ha sido desarrollada en el Laboratorio de Sistemas Digitales del Departamento de Ingeniería Eléctrica y de Computadoras de la Universidad Nacional del Sur y está basada en investigaciones realizadas durante el 1 de Abril de 2005 y el 31 de Marzo de 2007, bajo la dirección de los Doctores Javier Orozco y Ricardo Cayssials.

Como resultado de las investigaciones realizadas se han originado publicaciones de las cuales soy autor principal o coautor. Estas publicaciones y/o el contenido de esta tesis no han sido utilizados como contenido de otra tesis llevada a cabo en dicho ámbito. Las principales publicaciones en las que se basa la presente tesis son:

- **Duval**, M., Cayssials, R. and Orozco, J. "Real-Time Influences on Control Applications", 7º Simposio Argentino de Tecnología de Computación - AST 2006. Mendoza, 4 al 7 de septiembre de 2006. Páginas 256-268.

- Cayssials, R., **Duval**, M., Ferro E. and Alimenti. O. "uRT51: An Embedded Real-Time processor implemented on FPGA devices", Latin American Applied Research, ISSN 0327-0793, Indexada SCI, Vol.37, Nro.1, pp.:35-40, 2007.
- **Duval**, M., Cayssials, R., Orozco, J. y Ferro E. "An Adaptive Scheduling Mechanism for Real-Time Control Tasks". *Proc. Work-In-Progress (WIP) session of 19th Euromicro Conference on Real-Time Systems, '07,* Pisa, Italy July 4-6, 2007.
- **Duval**, M., Cayssials, R., Alimenti, O., Ferro, E. y Orozco, J. "Jitter-Aware Scheduler for Real-Time Control Applications". *Proc*. *XXXIII Conferencia Latinoamericana de Informática, CLEI 2007*, 9-12 Octubre, San José, Costa Rica.

Martín Leandro Duval

Bahía Blanca, 29 de Junio de 2012

UNIVERSIDAD NACIONAL DEL SUR
Secretaría General de Posgrado y Educación Continua

La presente tesis ha sido aprobada el 29/06/2012, mereciendo la calificación de 10 (diez).

Resumen

Los sistemas de tiempo real fueron, en un principio, concebidos para ser utilizados en aplicaciones críticas tales como sistemas espaciales, aviación, control de centrales, plantas industriales. En este tipo de aplicaciones, las decisiones que provienen de estos algoritmos de control deben ser aplicadas antes de un determinado instante de tiempo denominado *vencimiento*. Debido a que estas aplicaciones son consideradas críticas, las primeras investigaciones definieron a estos sistemas como de *tiempo real duro*, en los cuales ningún vencimiento puede ser perdido.

Generalmente, las condiciones para garantizar tiempo real duro son pesimistas y los sistemas resultan en consecuencia sobredimensionados. Esta capacidad de procesamiento ociosa fue utilizada para la ejecución de tareas denominadas *blandas*. Estas tareas son atendidas por el sistema una vez que las condiciones temporales duras están garantizadas.

Una de las mayores áreas de utilización de sistemas de tiempo real son las aplicaciones de control. Sin embargo, y debido a las discrepancias existentes entre la teoría de control y la de tiempo real, existen perturbaciones que los sistemas de tiempo real producen en las aplicaciones de control. Estas perturbaciones son debido a que en un sistema de tiempo real, las estrategias de control pueden ser relegadas del uso del procesador produciendo retardos cuando existen tareas de

mayor prioridad que requieren su utilización. Como estos retardos no son constantes de invocación a invocación de la tarea de control, se produce un *jitter* que no es tenido en cuenta por la teoría de control y que puede provocar efectos indeseables sobre la aplicación.

Por consiguiente, el modelo de tareas considerado por la teoría clásica de tiempo real no es enteramente adecuado para su utilización en sistemas de control. Esto se debe a que en ellos, no resulta suficiente observar el cumplimiento de los vencimientos ya que aparecen nuevos requerimientos temporales no contemplados por dichos modelos como son las fluctuaciones en los tiempos de actualización de parámetros de control y de las señales de comando, que provocan perturbaciones indeseadas en el sistema a controlar.

En este contexto, esta tesis propone la utilización de técnicas que admiten una violación acotada de los requerimientos en pos de garantizar su aplicabilidad a sistemas de control intolerantes a variaciones en los tiempos de activación y respuesta de tareas críticas aprovechando tanto los tiempos ociosos del sistema como produciendo pérdidas controladas de vencimientos que no comprometan la integridad del sistema.

El objetivo principal de esta tesis consiste en el análisis y desarrollo de disciplinas de planificación en tiempo real de bajo jitter para aplicaciones de propósito dedicado en sistemas de control

Índice

Agradecimientos ..... *ii*
Prefacio ..... *iv*
Resumen ..... *vii*
Índice ..... *ix*
Capítulo 1 ..... ***1***
Introducción ***1***
1.1 Aportes de esta tesis. 3
1.2 Trabajos previos 4
1.3 Organización de la Tesis 5
Capítulo 2 ..... ***7***
Sistemas de Control ***7***
2.1 Control Digital 9
2.2 Definición del período de muestreo. 10
2.3 Robustez y sensibilidad a las perturbaciones 11
2.4 Conclusión 12
Capítulo 3 ..... ***13***
Sistemas de Tiempo Real ***13***
3.1 Conceptos básicos. 13
3.2 Modelo de Tareas. 15
3.3 Diseño de Sistemas de Tiempo Real 15
3.4 Estados de las tareas. 16
3.5 Algoritmos de Planificación. 17
3.5.1 Factor de Utilización del Procesador. 19
3.5.2 Prioridades Fijas. RM 19
3.5.3 Menor Tiempo al Vencimiento. EDF 20
3.6 Conclusiones 21
Capítulo 4 ..... ***22***
Planificación de Tareas de Control ***22***
4.1 Modelo de tareas de control 22
4.2 Caso A. Jitter de entrada y de salida. 24

4.3 Caso B. Solución al jitter de entrada. 25
4.4 Caso C. Solución al jitter de entrada y de salida, con retardo en el cálculo de control. 25
4.5 Caso D. Solución al jitter de entrada y de salida, con menor retardo en el cálculo de control. 26
4.6 Conclusiones 27
Capítulo 5 28
Métodos Flexibles en Sistemas de Tiempo Real 28
5.1 Control realimentado elástico 28
5.2 Controladores de modelo predictivo y Compensación de jitter 29
5.3 Restricciones temporales de tareas de control. 30
5.4 Planificación de tiempo real débil-duro (weakly-hard). 30
5.5 Conclusiones 31
Capítulo 6 32
Análisis de Perturbaciones 32
6.1 Introducción 32
6.2 Caso de Estudio: El Péndulo Invertido. 33
6.3 Experiencias realizadas. 36
6.3.1 Experimento 1. Selección del período de muestreo de la tarea de control. . 37
6.3.2 Experimento 2. Análisis de las perturbaciones del jitter. 38
6.3.3 Experimento 3. Variación el período de la tarea en tiempo de ejecución.... 39
6.3.4 Experimento 4: Salteando invocaciones de la tarea. 40
6.4 Análisis de resultados. 41
6.5 Conclusiones. 41
Capítulo 7 43
Planificador de Bajo Jitter. Un Mecanismo Flexible de Planificación de Tareas de Control de Tiempo Real 43
7.1 Introducción 44
7.1.1 Planificador Dual- Priority 45
7.2 Planificador de Bajo Jitter: PBJ 45
7.3 Un caso de estudio. El Péndulo Invertido 50
7.4 Experiencias realizadas 51

7.5 Análisis y resultados 54
7.6 Conclusiones 56
Capítulo 8 57
Utilización del uRT51 para Planificación de Tareas de Control 57
8.1 Introducción 57
8.2 La arquitectura del uRT51 59
8.2.1 Tiempo y Eventos en el uRT51. 60
8.2.2 Tareas y Prioridades. 61
8.2.3 Configuración de Tiempo Real. 62
8.3 Plataforma de desarrollo del uRT51 63
8.4 Experiencias Realizadas 63
8.4.1 Caso de Estudio: Control de velocidad de un motor de DC 64
8.4.2 Implementación y Evaluación. 66
8.5 Análisis y resultados 68
8.6 Conclusiones. 69
Capítulo 9 71
Control de Sistemas Complejos con Mecanismos Flexibles 71
9.1 Introducción. 71
9.2 Variación del período de muestreo. 72
9.3 Implementación del mecanismo propuesto 73
9.4 Caso de estudio. Sistema dos masas y resorte. 75
9.5 Experiencias realizadas. 76
9.6 Análisis y resultados 77
9.7 Conclusiones 78
Capítulo 10 79
Conclusiones y Futuros Trabajos 79
Apéndice A 82
Símbolos 82
Apéndice B 84
TrueTime 84
B.1 Introducción. 84
B.2 El Simulador. 85
B.2.1 El Bloque de Computadora 86
B.2.2 El Bloque de Red. 87

B.2.3 Inicialización. 88
B.3 Conclusiones. 89
Referencias ***90***

Capítulo 1

Introducción

Los sistemas de control consisten básicamente en una planta o proceso y un controlador. La planta es el sistema donde conviven las señales o variables que se desean controlar. El controlador generalmente se compone de tres componentes que realizan tres funciones bien diferenciadas; los sensores estratégicamente ubicados, leen la información de las señales relevantes de la planta; el controlador propiamente dicho, calcula las acciones de control y el actuador inyecta estas acciones de control en la planta para alcanzar la eficiencia de control deseada. En teoría de control digital se asume que los sensores son muestreados periódicamente y las salidas del sistema son actualizadas tan pronto como la información esté disponible en ellos, o al menos con un retardo constante.

Un sistema de control típico está constituido de varias de esas actividades de control en adición a otras operaciones que no lo son (tales como tareas de monitoreo, interacción con usuarios, entre otras). Estas operaciones de control necesitan ser ejecutadas periódicamente y dentro de un estricto límite de tiempo para garantizar que el sistema cumpla con los requerimientos de desempeño deseados.

Existen técnicas de modelado que se aplican para obtener una expresión matemática de las características de control de la aplicación. Desde el modelo de control resultante, se puede diseñar el controlador que tendrá la misión de hacer que la aplicación tome un comportamiento controlado.

Con estas consideraciones podemos afirmar que un sistema de control es un sistema de tiempo real. Actividades concurrentes múltiples se deben realizar dentro de un vencimiento bien definido. No obstante, hay discrepancias importantes entre las hipótesis y los modelos computacionales usados en tales disciplinas. La más obvia es la interpretación del período y el vencimiento. En sistemas de tiempo real, la noción de tarea periódica implica una computación que comienza cada kT unidades de tiempo, donde T es el período y $k = 0, 1, 2, \ldots$, y tiene que ser completada antes de D unidades desde su comienzo (en la mayoría de los casos D es considerado igual al período). En teoría de control, "periodicidad" significa un cómputo que es realizado con una separación de *exactamente* T unidades. El modelo de tiempo real no considera el hecho de que el tiempo de comienzo es esencialmente el mismo que el considerado desde la perspectiva de la teoría de control. En el caso ideal, se asume que el sensado, la computación y la actuación no consumen tiempo y por lo tanto la tarea de control es ejecutada exactamente a kT instantes de tiempo. Otro ejemplo es el hecho de que los sistemas de control son robustos y pueden tolerar la pérdida *ocasional* de vencimientos, de todos modos tal concepto de pérdida de vencimientos no es bien expresado en el dominio de tiempo real.

El tiempo que una tarea de control tiene que esperar para ser ejecutada depende del tiempo de computación requerido por tareas de mayor prioridad listas para ser ejecutadas. Debido a que el patrón con que las tareas solicitan ser ejecutadas no es fijo, esto resulta en una interferencia variable y por lo tanto en un tiempo de respuesta variable de la tarea. Esta variación de tiempo, denominado *jitter*, no es modelado por la teoría de control y consecuentemente puede producir efectos indeseados en la aplicación de control

La solución trivial para reducir el jitter es asignar a las tareas de control las mayores prioridades. Sin embargo, como un sistema de tiempo real debe ejecutar más de una tarea de control además de otras tareas importantes (como tareas de

seguridad), esto no es posible. Las tareas de mayor prioridad tendrán bajo jitter mientras que las tareas de menor prioridad tendrán jitter más alto acarreando un mal desempeño.

El objetivo principal de esta tesis consiste en el análisis y desarrollo de disciplinas de planificación en tiempo real de bajo jitter para aplicaciones de propósito dedicado en sistemas de control. La reducción del jitter, permitirá disminuir los efectos adversos que éste produce sobre las aplicaciones (inestabilidad, mayor consumo de energía, ruido, vibraciones, interferencias). El logro de este objetivo requerirá la utilización de plataformas para la implementación y simulación de disciplinas de tiempo real flexibles sobre aplicaciones de control.

Para la simulación de los algoritmos de diagramación en aplicaciones de control se utilizó la herramienta TrueTime desarrollada por Henriksson ([25]). Esta herramienta permite la simulación en un ambiente de Simulink/Matlab del conjunto tiempo real/control para verificar las perturbaciones que diferentes algoritmos de diagramación producen en las aplicaciones de control.

Para la implementación de los algoritmos propuestos y las mediciones sobre una plataforma real se desarrollaron técnicas de HW/SW codesign sobre dispositivos lógicos programables (FPGA). Mediante estas técnicas es posible utilizar una arquitectura de procesamiento específicamente diseñada para tiempo real. De esta manera, se podrán implementar, en futuros trabajos, plataformas de procesamiento de tiempo real adecuadas para sistemas de propósito dedicado (*embedded systems*) para aplicaciones de control en donde las restricciones temporales deben ser satisfechas para lograr una eficiencia satisfactoria, al mismo tiempo que otras restricciones de diseño podrán ser más fácilmente cumplidas.

1.1 Aportes de esta tesis.

Los aportes originales de esta tesis están enfocados en la obtención de sistemas de tiempo real más eficientes y económicos que permitirán su utilización en las más diversas aplicaciones. Esto repercutirá favorablemente en dos aspectos:

- Obtención de sistemas más económicos para su utilización masiva. La reducción en la complejidad y costo de los sistemas de tiempo real permitirá la realización de instrumental, control de aplicaciones y dispositivos microinformáticos baratos. De esta manera podrán ser utilizados por sectores sociales que actualmente no pueden acceder a estos dispositivos por su capacidad económica.
- Implementación de sistemas microinformáticos en aplicaciones que actualmente prescinden de ellos por los elevados costos actuales. La automatización hogareña y el control de energía son ejemplos en donde la microinformática permitiría una mejor calidad de vida y un mejor aprovechamiento de la energía si los costos de la implementación fueran adecuados.

Por estos objetivos, la implementación de sistemas de tiempo real de bajo costo permitiría su utilización en áreas donde actualmente no son económicamente rentables y al mismo tiempo posibilitaría un mejor diseño en las actuales aplicaciones de los sistemas de tiempo real.

1.2 Trabajos previos

Varios trabajos han analizado los efectos que producen la planificación de tareas en tiempo real sobre aplicaciones de control. En [2], se propone un algoritmo de asignación de prioridades que tiene en cuenta las características de control para maximizar el desempeño de control. El "Control Server" es presentado en [16] para la implementación de tareas de control en sistemas de tiempo real flexibles. En [4], es propuesto un modelo para las tareas de control que las separa en tres subtareas con diferentes características de tiempo real. En [23], se presenta una estrategia de planificación realimentada para múltiples tareas de control que optimiza el desempeño total de control, conforme a limitaciones de carga del sistema. En [47], se analizan los efectos sobre una aplicación de control por medio de simulaciones usando modelos de control simples. En [42], se analizan las perturbaciones que un sistema de tiempo real produce sobre una aplicación de control. Todos estos trabajos proponen diferentes modelos de tareas

de control para reducir el jitter, pero ninguno de ellos trata además de mejorar el tiempo de respuesta de las tareas que no son de control, cuestión que se logra con uno de los mecanismos desarrollados en esta tesis. Además, muchos de estos trabajos realizan simulaciones sobre sistemas simples. En el desarrollo de esta tesis se demostrará que hay consideraciones prácticas que vuelven estos mecanismos no adecuados.

Por otro lado, tanto en [23] como en [14], se proponen algoritmos adaptivos on-line para cambiar el período de la tarea de acuerdo a la carga del sistema. Sin embargo, no se considera la degradación que producen sobre el desempeño de control.

En [33], se propone un mecanismo que modifica la estrategia de control en tiempo de ejecución para compensar los efectos del jitter. En dicho trabajo no son tenidos en cuenta los efectos variables del muestreo de la entrada debido a que el mecanismo no modela la discretización de la señal de entrada. Cuando el sistema es implementado, la entrada ya no es continua y la modificación del período introduce un error en el muestreo que no es considerado en dicho trabajo.

1.3 Organización de la Tesis

Esta tesis está organizada de la siguiente manera. En el Capítulo 2, se desarrollan los conceptos básicos de sistemas de control donde se destacan definiciones como las del período de muestreo, robustez y sensibilidad a las perturbaciones que serán usadas a lo largo de toda la tesis. En el Capítulo 3, se describen los términos y definiciones utilizadas en tiempo real. En el Capítulo 4, se consideran los diferentes modelos de implementación de tareas de control en sistemas de tiempo real y se analizan los diferentes efectos sobre los mismos. En el Capítulo 5, se introducen algunos métodos flexibles de planificación aplicados a la teoría de control. En el Capítulo 6, se analizan los efectos que los sistemas de tiempo real producen sobre las aplicaciones de control. Usando los resultados obtenidos en el Capítulo 6, en el Capítulo 7 se presenta un mecanismo flexible de planificación de tareas de control denominado Planificador de Bajo Jitter (PBJ) que mejora la eficiencia tanto de tareas de control como de tareas que no son de

control. En el Capítulo 8, se describen y evalúan las principales características del procesador uRT51, diseñado para aplicaciones embebidas de control de tiempo real. Nuevamente, usando los resultados obtenidos en el Capítulo 6, se presenta en el Capítulo 9 un nuevo mecanismo flexible para control de sistemas complejos. En el Capítulo 10, se exponen las conclusiones y se proponen los futuros trabajos. En el Apéndice A, se enumeran todos los símbolos utilizados. El Apéndice B, se presenta la herramienta TrueTime usada en esta tesis para realizar las simulaciones. En la última Sección, se muestran las referencias utilizadas.

Capítulo 2

Sistemas de Control

En esta sección se presentan los principales conceptos usados en modelado de sistemas de control. De este desarrollo, se establecerá una relación con las características temporales de un sistema de tiempo real.

Un sistema de control básico consiste generalmente de una planta y de un controlador. La planta es el sistema en el cual se encuentra las señales o variables que se desean controlar. El controlador es el que realiza este trabajo en tres funciones bien diferenciadas: primero lee la información que le proporcionan dichas variables por medio de sensores ubicados estratégicamente en la planta, luego calcula las acciones de control y por último actúa sobre la planta para que el sistema alcance la eficiencia de control especificada.

Para expresar matemáticamente la dinámica tanto de la planta como del controlador se aplican técnicas de modelado. El modelo de un sistema es una abstracción simplificada usada para predecir el comportamiento del mismo. Es frecuente obtener un modelo analítico del sistema usando leyes que determinan su comportamiento

El modelo de *espacio de estado* es una técnica muy usada en modelado de sistemas. En un sistema de tiempo continuo, un modelo en el espacio de estados consiste de un vector de ecuaciones diferenciales de primer orden, denominado *ecuación de espacio de estado* y una ecuación de salida, las cuales son mostradas en las ecuaciones (1) y (2) respectivamente.

$$\dot{x}(t) = f(x(t), u(t), t) \tag{1}$$

$$y(t) = g(x(t), u(t), t) \tag{2}$$

Donde $x(t)$ es denominada *variable de estado*, $u(t)$ es la *variable de entrada,* $y(t)$ es la *variable de salida,* **f**() es la *función de estados* y **g**() es la *función de salida*.

Un prototipo típico generalmente encontrado en la teoría clásica de control se muestra en la Figura **1**. En esta figura, la señal realimentada $\boldsymbol{y}(t)$ es la *señal de salida* del sistema, que es comparada con la *señal de referencia de entrada* $r(t)$, para obtener la *señal error* $e(t)$ que es la que entrará al controlador para luego obtener la *señal de control* $u(t)$ que será inyectada en la planta.

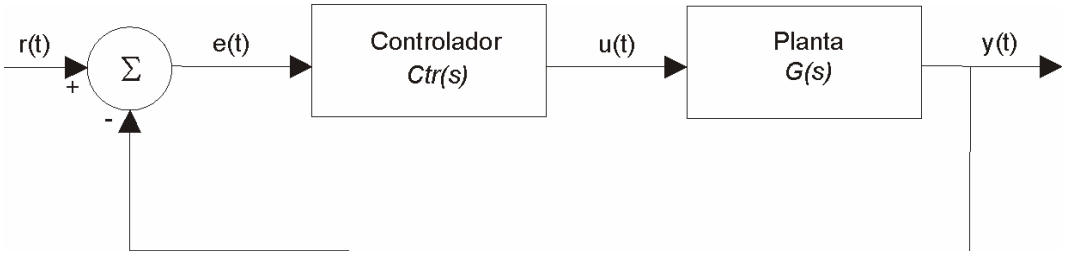

Figura 1. Sistema de control continuo en el tiempo.

El problema de diseño es especificar la función de transferencia $Ctr(s)$ para cumplir con las características deseadas del sistema a lazo cerrado. Dentro de las características más comunes a tener en cuenta en el diseño, se encuentran la estabilidad y algunas especificaciones para la respuesta al escalón como el porcentaje de sobrepico, el tiempo de establecimiento y el error en estado estacionario.

Dos de las propiedades más importantes de un sistema de control son la linealidad y la invariancia en el tiempo ([43]). El modelo de un sistema lineal

invariante y continuo en el tiempo es expresado en espacio de estado en la ecuación (33).

$$\begin{aligned} \dot{x}(t) &= \mathbf{A} \cdot x(t) + \mathbf{B} \cdot u(t) \\ y(t) &= \mathbf{c} \cdot x(t) + \mathbf{d} \cdot u(t) \end{aligned} \tag{3}$$

Donde **A** es la *matriz del sistema,* **B** es la *matriz de entrada*, **c** es la *matriz de salida* y **d** es la *matriz de realimentación*.

2.1 Control Digital

Cuando un controlador es implementado sobre una computadora, se debe usar un modelo discreto. La Figura 2 muestra una planta en tiempo continuo controlada por un controlador en tiempo discreto.

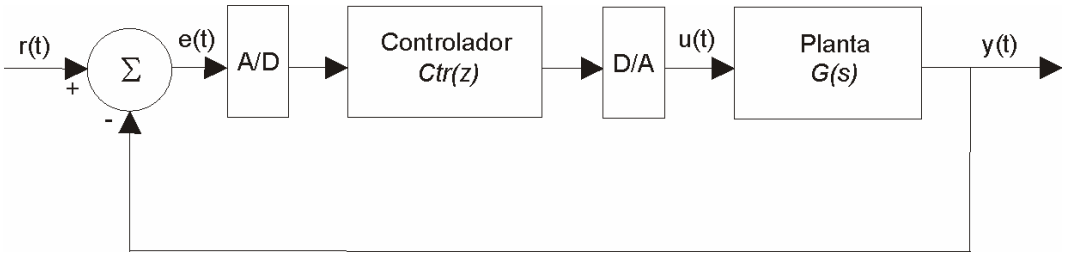

Figura 2. Sistema de control discreto en el tiempo.

La función de tranferencia del controlador *Ctr*(*z*) es implementada como software y ejecutada por un procesador. La entrada a *Ctr*(*z*) es una secuencia de números obtenido desde un convensor A/D y la salida es otra secuencia de números que es convertida a una señal de control contínua a tramos a través de un conversor D/A.

El modelo de espacio de estados de un sistema de tiempo discreto es expresado por las ecuaciones a diferencia mostradas en la ecuación (4).

$$\begin{aligned} x[k+1] &= \mathbf{\Phi} \cdot x[k] + \mathbf{\Gamma} \cdot u[k] \\ y[k] &= \mathbf{c} \cdot x[k] + \mathbf{d} \cdot u[k] \end{aligned} \tag{4}$$

En [43], se muestra que dado un modelo en espacio de estados de una planta caracterizada por las matrices **A, B, c,** y **d** como el representado en la ecuación (3)

y un intervalo de muestreo T, el sistema equivalente en tiempo discreto está dado por las matrices **Φ, Γ, c** y **d,** donde

$$\mathbf{\Phi} = e^{\mathbf{A}\cdot T}, \quad \mathbf{\Gamma} = \int_0^T e^{\mathbf{A}\cdot\tau}\cdot\mathbf{B}\cdot d\tau \tag{5}$$

Este modelo discreto obtenido a través de esta transformación no es una aproximación, sino una descripción exacta del comportamiento de la planta en cada instante de muestreo T.

Variando el intervalo de muestreo a lo largo del tiempo, el modelo de tiempo discreto deja de ser válido, y por consiguiente mismas acciones producirán efectos diferentes. Un retardo variable durante la actualización de las salidas, producirá un efecto que puede no ser modelado conduciendo a un comportamiento no deseado en el sistema. El objetivo de esta tesis es obtener mecanismos que reduzcan estos comportamientos no deseados en el sistema.

2.2 Definición del período de muestreo.

En control digital la elección del período de muestreo es muy importante. Si se elige muy grande, la señal de tiempo continuo no podrá ser reconstruida. Por otro lado, si se elige muy pequeño, la carga computacional aumentará y posiblemente la actualización de la salida no tendrá lugar.

Existen varias reglas empíricas para determinar un rango de selección para la frecuencia de muestreo, w_s (definida en la ecuación (**6**)) y por consiguiente también para el período de muestreo ($T = 1/w_s$).

$$w_s = 2\cdot\frac{\pi}{T}\left(rad/seg\right) \tag{6}$$

Muchas de esas reglas son basadas en la relación entre la frecuencia de muestreo y el ancho de banda del sistema a lazo cerrado, denominado w_B.

De acuerdo a las reglas en [43], w_s puede ser elegido de la siguiente manera:

- Si la planta (o los sensores) están sujetos a perturbaciones aleatorias cuyo contenido de frecuencia excede el ancho de banda del sistema de control, se elige: $20 \leq w_s/w_B \leq 40$.
- Si la planta no esta sujeta a perturbaciones aleatorias, se elige: $5 \leq w_s/w_B \leq 10$

Una velocidad de muestreo alta puede hacer que el sistema se torne muy sensible a la precisión de los parámetros y a los errores de cuantización.

2.3 Robustez y sensibilidad a las perturbaciones

En la implementación de sistemas de control, su comportamiento puede diferir del esperado debido a varias cuestiones de implementación. Algunos de estos factores son enumerados a continuación:

a. Variaciones de los parámetros del sistema. Tanto las tolerancias mecánicas como la calidad del producto, pueden producir diferentes características de funcionamiento en la implementación final. Por lo tanto, un sistema de control debe ser lo suficientemente robusto para tener en cuenta todas estas variaciones.
b. Precisión e histéresis de sensores y actuadores. Los sensores y actuadores reales pueden no responder a pequeñas variaciones en sus entradas debido a propiedades de precisión, sensibilidad o histéresis.
c. Cuantificación de errores de conversores. Conversores A/D y D/A tienen cuantificación de errores debido a la longitud finita de sus representaciones binarias.
d. Rango dinámico finito de sensores y actuadores. La respuesta tanto de un sensor como de un actuador es limitada a un cierto rango y debe ser tenido en cuenta en análisis de estabilidad y robustez.

Mucho de estos factores pueden ser reducidos mejorando el hardware utilizado, con la consecuencia del aumento del costo. En este caso se debe tener en cuenta una solución de compromiso.

2.4 Conclusión

En esta sección se describieron los principales conceptos en que se basa un diseño de un sistema de control. En el resto de la tesis se analizarán las diferentes perturbaciones que los sistemas de tiempo real introducen en estos modelos y propondremos diferentes mecanismos para disminuir dichas perturbaciones usando la teoría de tiempo real.

Capítulo 3

Sistemas de Tiempo Real

En este capítulo se presentan los principales conceptos y se introducen los términos y definiciones usadas en la teoría de tiempo real.

3.1 Conceptos básicos.

Se denominan Sistemas de Tiempo Real a aquellos sistemas informáticos cuyos aspectos temporales son parte de su especificación. La correctitud de estos sistemas no depende únicamente de su correcto comportamiento aritmético-lógico, sino también del instante de tiempo en que se presenten los resultados producidos. Esto ocurre comúnmente en sistemas que deben interactuar con un entorno que se modifica en todo momento, como es el caso de sistemas de control de procesos, robótica, telecomunicaciones, multimedia, redes de sensores, entre otros.

Las aplicaciones de tiempo real están compuestas por un conjunto de tareas, donde cada una cumple una determinada función en la aplicación ([9]). Éstas se

componen de una serie infinita de instancias de sí misma. Cada instancia es independiente respecto de la anterior ([12]).

La característica principal de estos sistemas, es la permanente supervisión de los datos de entrada, para reaccionar en consecuencia cuando correspondiere. Esta reacción debe finalizar antes de un determinado tiempo, denominado *vencimiento*. Las consecuencias que pueden provocar el no cumplimiento de algunos vencimientos, divide a los sistemas de tiempo real en *duros* y *blandos*.

Se dice que un sistema de tiempo real es duro si la pérdida del vencimiento de alguna de sus tareas, puede producir consecuencias catastróficas sobre el ambiente al cual controlan. Para alcanzar factibilidad el sistema debe, a priori, garantizar a cada tarea que todas sus instancias pueden ser ejecutadas completamente dentro de sus vencimientos.

Por otro lado, se dice que un sistema de tiempo real es blando, si la pérdida del vencimiento de alguna de sus tareas no afecta el desempeño general del sistema. Aún en esta situación, el resultado logrado resulta útil respecto de la aplicación.

Como un buen ejemplo de sistemas de tiempo real blando puede citarse la transmisión de multimedia, en la cual la pérdida ocasional de algún cuadro de video, si bien es no deseable, no trae aparejada mayores consecuencias. Otro ejemplo muy interesante de estos sistemas es la aplicación de tiempo real en una red de sensores ([35]).

El desarrollo de un sistema de tiempo real, como cualquier otro sistema, posee dos etapas definidas: análisis y diseño. En la etapa de análisis, se especifica todo el comportamiento del sistema de tiempo real. Se define el modelo de las tareas imponiendo las restricciones funcionales y temporales del sistema con el medio que lo rodea. En la etapa de diseño, se determina la forma en que será organizado el sistema, para cumplir las especificaciones y para su posterior implementación.

Ambas etapas no deben ser secuenciales o excluyentes, sino que debe existir una permanente interacción entre el analista y el diseñador para mejorar tanto las especificaciones como la implementación final.

3.2 Modelo de Tareas.

Una tarea es un conjunto de operaciones secuenciales que tienen como objetivo cumplir con alguna de las especificaciones funcionales determinadas para el sistema. Existen dos tipos de tareas, *tareas periódicas* y *tareas no-periódicas.*

Una *tarea periódica* consiste en una computación que es ejecutada repetitivamente con un patrón regular y cíclico. La duración del intervalo entre dos solicitudes de ejecución es denominada *período*.

Una *tarea no-periódica*, es una computación que se ejecuta como respuesta a un evento asincrónico. Es importante determinar para una tarea no-periódica el mínimo intervalo que puede existir entre dos eventos consecutivos o *mínimo tiempo entre arribos*. Este parámetro determinará la mínima separación entre dos solicitudes de ejecución consecutivas de dicha tarea.

Con las consideraciones realizadas, el modelo general de especificación de un sistema de tiempo real consiste en un conjunto **Π** de *m* tareas periódicas y no periódicas el cual es mostrado en la ecuación (**7**).

$$\mathbf{\Pi}(m) = \{ \tau_i = (T_i, D_i) \mid 1 \leq i \leq m \} \qquad (7)$$

Cada tarea τ_i, es caracterizada por su *período*, (en el caso de tareas periódicas) o su *mínimo tiempo entre arribos*, (en el caso de tareas no periódicas), T_i; y su *vencimiento,* D_i.

3.3 Diseño de Sistemas de Tiempo Real

Como en la mayoría de los procesos de diseño, el diseñador debe enfrentarse a situaciones de compromiso.

Las tareas son las encargadas de relacionar todos los componentes del sistema para su correcto funcionamiento. Cuando ocurren eventos provocados por el ambiente o producidos internamente en el sistema, las tareas deben reaccionar para actuar en consecuencia antes de su respectivo vencimiento. En el diseño, las tareas estarán conformadas por software ejecutable en el hardware del sistema.

Puesto que las tareas deben ser ejecutadas en un hardware con velocidad de procesamiento finita, cada una de ellas tendrá un tiempo de ejecución asociado. El *máximo tiempo de ejecución* de cada tarea, C, es un parámetro importante en el diseño de sistemas de tiempo real. De esta manera, el conjunto de m tareas, $\mathbf{\Pi}(m)$ queda especificado de la siguiente manera para el diseñador:

$$\mathbf{\Pi}(m) = \{ \tau_i = (T_i, C_i, D_i) \mid 1 \leq i \leq m \} \quad (8)$$

Existen otros parámetros de la tarea tales como la cantidad de memoria del sistema que requiere, los recursos que necesita y características especiales del hardware. Sin embargo, para la verificación de las especificaciones temporales, generalmente son sólo necesarios el período, el máximo tiempo de ejecución y el vencimiento de cada una de las tareas, de los cuales el período y el vencimiento dependen de las especificaciones de tiempo real descriptas en la etapa de Análisis y el máximo tiempo de ejecución dependerá de la etapa de Diseño.

3.4 Estados de las tareas.

Mientras esté activo un sistema de tiempo real, las tareas que lo componen pueden estar en uno de los siguientes estados:

- Ejecución: la tarea está siendo ejecutada por el procesador. Puede existir una sola tarea en este estado por unidad procesadora en el sistema.
- Espera: la tarea está esperando un evento para pedir ser ejecutada por el procesador. El evento puede provenir de un dispositivo de entrada/salida, un temporizador del sistema, de otra tarea o un evento interno del sistema. Pueden existir varias tareas en este estado en un instante determinado.
- Lista: la tarea está solicitando el procesador para ser ejecutada. Al igual que el estado de espera, pueden existir varias tareas en el estado de lista en un instante determinado.

Los nombres y cantidad de los estados pueden variar según el sistema operativo del que se tratare. La Figura 3 presenta un diagrama típico de estados de las tareas de un sistema y los diferentes eventos que producen las transiciones en los estados.

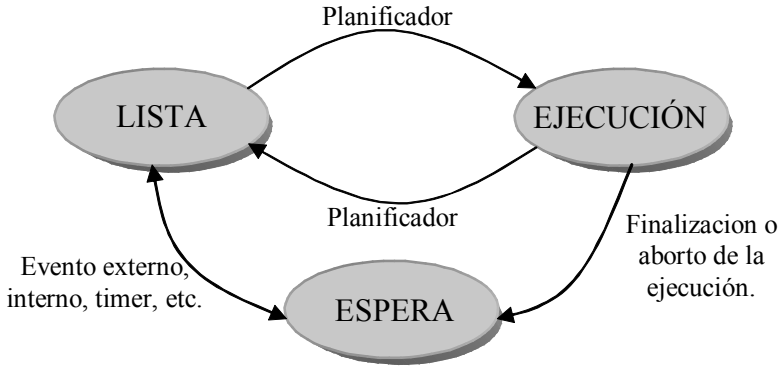

Figura 3. Planificador de estados de las tareas

Las transiciones entre los posibles estados de una tarea se deben a diferentes causas. Del estado de espera al de lista, es necesario que ocurra un evento para que suceda la transición. Cuando esta transición ocurre, se dice que la tarea *arriba* o se *genera*. La transición del estado de ejecución al de espera ocurre cuando la tarea ha terminado de ejecutarse y ejecuta la primitiva "exit" del Sistema Operativo, para indicarle que ya no requiere más la utilización del procesador. Las transiciones del estado de lista al de ejecución y viceversa son generalmente realizadas por una tarea del Sistema Operativo denominada *planificador* de tareas. El planificador selecciona la tarea que pasará al estado de ejecución entre todas las tareas que están en el estado de lista y eventualmente pasa al estado de lista la tarea que está en estado de ejecución. Puede existir, según el sistema de tiempo real del que se tratare, una transición que lleve una tarea del estado de lista al de espera, pero no se la considerará por no influir en los resultados de tiempo real de esta tesis.

3.5 Algoritmos de Planificación.

Según lo desarrollado por Liu & Layland en [30], un algoritmo de planificación, o simplemente planificador, es un conjunto de reglas que determinan cuál es la tarea a ser asignada a un recurso o procesador en un determinado instante de tiempo.

Los algoritmos de planificación pueden ser divididos en dos tipos bien diferenciados, los *Estáticos o Fijos* y los *Dinámicos.* En todos éstos, la noción de

prioridad (Fija o Dinámica) es utilizada para ordenar el acceso al procesador. Cuando hay varias tareas que quieren acceder a un recurso, ésto se resuelve asignando el mismo a la tarea con la más alta prioridad ([38]). Cada vez que una tarea requiere del recurso compartido, se dice que la tarea es *invocada*.

Existen condiciones exactas, necesarias y suficientes para el análisis de la planificabilidad. Éstas pretenden garantizar los requerimientos temporales respetando el acceso compartido al procesador.

Los algoritmos de planificación estáticos son aquellos en los que la asignación de prioridades a las tareas se realiza sólo una sola vez y se mantienen a lo largo del tiempo. También se los llama Algoritmos de Planificación de Prioridades Fijas. Por otro lado, los Algoritmos de Planificación Dinámicos son aquellos en los que las prioridades de las tareas pueden cambiar de período a período.

En [30], se presentan dos algoritmos muy usados y conocidos en la teoría de tiempo real como lo son el algoritmo estático FP (Fixed Priority) y algoritmo dinámico EDF (Earliest Deadline First), que serán utilizados en esta Tesis para comparar sus desempeños con los de los mecanismos propuestos.

Un parámetro muy importante para medir la eficiencia de los planificadores es el *tiempo de respuesta*. Éste es definido como el tiempo entre el pedido de ejecución de una tarea y la finalización de la ejecución de la misma.

En particular, Liu & Layland presenta una serie de suposiciones respecto al entorno, para el adecuado funcionamiento de las políticas que plantea:

- Conjunto fijo de tareas periódicas.
- El vencimiento de cada tarea es igual a su período ($T = D$).
- Tareas independientes, esto es, no tienen relaciones de precedencia o de recursos.
- Todas las tareas deben terminar de ejecutarse antes de su próxima instancia.
- Ninguna tarea puede desalojarse a sí misma.
- Todas las tareas son completamente desalojadas.
- Los tiempos de ejecución de cada tarea son fijos (C =*constante*).

- El único recurso en común es el procesador.

3.5.1 Factor de Utilización del Procesador.

Se define Factor de Utilización ([30]) a la fracción de tiempo de procesamiento del procesador que necesita una tarea para ser ejecutada. Por consiguiente, el factor de utilización de una tarea τ_i, denominado U_i, queda definido como:

$$U_i = \frac{C_i}{T_i} \qquad (9)$$

Si consideramos el caso en que $D_i = T_i$ y dado que C_i / T_i es la fracción de tiempo del procesador utilizado en ejecutar la tarea τ_i, en un conjunto de m tareas, el Factor de Utilización Total del sistema, denominado U, es:

$$U = \sum_{i=1}^{m} \left(C_i / T_i \right) \qquad (10)$$

Éste indica la carga del sistema. Si dicho valor supera la unidad se dice que el sistema está *sobrecargado*. Esto implica que existirán tareas que no podrán satisfacer su ejecución antes de su vencimiento, debido a que la capacidad del procesador es inferior al requerimiento del sistema. Este es el límite superior para satisfacer el requerimiento de que todas las tareas cumplan sus vencimientos.

Cuando el factor de utilización es igual a la unidad, el sistema se denomina *saturado*. En cambio, cuando este número no alcanza la unidad se dice que el sistema está *relajado*. Sin embargo, sistemas relajados o saturados pueden perder vencimientos debido a la política de planificación.

3.5.2 Prioridades Fijas. FP

Bajo las suposiciones realizadas anteriormente, Liu y Layland, en [30], proponen un método de asignación de prioridades para la disciplina de prioridades fijas. Este método, denominado *Periodos Monotónicos Crecientes* (RM), es basado en la tasa de arribo de los pedidos de ejecución. A tareas con menor período, se les asigna las mayores prioridades.

Esta asignación de prioridades es óptima en el sentido de que ninguna otra regla de asignación de prioridades fijas puede planificar un conjunto de tareas que no puede ser planificado bajo RM. Es decir, si un conjunto de tareas no puede ser planificado bajo RM, entonces no puede ser planificado por ningún otro algoritmo de prioridades fijas.

En [30], se obtiene una cota superior para analizar la planificabilidad de estos algoritmos de prioridades fijas:

$$U_p = m\left(2^{\frac{1}{m}} - 1\right) \qquad (\mathbf{11})$$

Teniendo en cuenta la ecuación (10), un conjunto de *m* tareas es planificable bajo RM si se cumple:

$$U \leq U_p \qquad (\mathbf{12})$$

Esta condición es suficiente y no necesaria. Esta condición de diagramabilidad es muy conservadora y pueden existir sistemas planificables usando prioridades fijas que no son cubiertos por esta cota.

3.5.3 Menor Tiempo al Vencimiento. EDF

En [30], Liu y Layland, plantean una asignación de prioridades dinámicas basadas en los vencimientos de las tareas. A la tarea que en su instancia actual tiene el menor vencimiento, se le asigna la mayor prioridad. Por otro lado, la tarea que en su instancia actual tiene el vencimiento más lejano, se le asigna la menor prioridad.

Este método es óptimo en el sentido de que, si un sistema no es planificable por EDF (Menor Tiempo al Vencimiento), no lo será por ninguna otra disciplina de prioridades.

Teniendo en cuenta la ecuación (10), un conjunto de tareas es planificable bajo EDF sí y sólo sí:

$$U \leq 1 \qquad (\mathbf{13})$$

Esta es una condición necesaria y suficiente para garantizar la diagramabilidad de un sistema de tiempo real diagramado mediante una disciplina de prioridades EDF.

3.6 Conclusiones

En este capítulo se presentaron los principales conceptos de tiempo real que serán utilizados en los próximos capítulos de la tesis. Para ésto, se hizo referencia al trabajo más conocido de tiempo real como es el de Liu & Layland ([30]).

También se presentaron los algoritmos de planificación más relevantes de prioridades fijas y dinámicas como lo son RM y EDF. En esta tesis, para mostrar las bondades de los mecanismos que se desarrollarán en futuros capítulos, se comparará la eficiencia de los mismos con la de estos algoritmos de planificación.

Capítulo 4

Planificación de Tareas de Control

En esta sección, se consideran los diferentes modelos de implementación de tareas de control en sistemas de tiempo real y se analizan sus diferentes características.

4.1 Modelo de tareas de control

Un sistema de tiempo real para control digital incluye además de tareas de control, otras que no lo son. Existen tareas de seguridad además de otras funciones secundarias (por ej.: tareas de comunicación y de interfase entre hombre-máquina) que deben ser ejecutadas por el sistema de tiempo real. Cuando una tarea de mayor prioridad requiere ser ejecutada, ésta desaloja del procesador a la que está siendo ejecutada. De esta manera, decimos que la tarea de menor prioridad fue *apropiada del uso del procesador*, o simplemente *apropiada*, y tal apropiación puede producir perturbaciones en la ejecución de las tareas de control. Alguna de las perturbaciones estudiadas aquí, pueden deberse a la ejecución de esta clase de tareas.

De acuerdo a las funciones que un controlador debe realizar, podemos dividir a la tarea de control en tres subtareas con diferentes requerimientos de tiempo real:

- **Subtarea de muestreo**: ésta es ejecutada periódicamente y es la encargada de leer las entradas del sistema. Su ejecución tiene que ser estrictamente periódica ($D << T$) para evitar no linealidades y variancias en el tiempo que pueda conducir a dinámicas incontrolables. El período de la subtarea de muestreo es definido por el intervalo de muestreo del modelo de control de tiempo discreto.
- **Subtarea de cálculo**: ésta es ejecutada después de que su correspondiente subtarea de muestreo ha finalizado y es la encargada de computar la estrategia de control. Su tiempo de ejecución depende de la complejidad de la estrategia de control implementada.
- **Subtarea de actuación**: ésta puede ser ejecutada cuando la subtarea de cálculo ha finalizado o después de un tiempo fijo desde la invocación de la subtarea de muestreo. Ésta debe ser ejecutada tan pronto como haya sido invocada.

Estas subtareas pueden ser implementadas como diferentes tareas (threads) o como una tarea monolítica entera. Las tareas de control pueden ser ejecutadas de acuerdo a diferentes disciplinas de prioridad pero debe tenerse cuidado en cuanto a la apropiación de las subtareas de muestreo. Una apropiación de subtarea de muestreo no está permitida cuando ésta usa un conversor A/D que es compartido entre diferentes subtareas de muestreo.

El conversor A/D es un circuito electrónico sensible que toma un intervalo no despreciable para convertir una señal analógica en su representación binaria. El hecho de compartir el conversor A/D, puede producir implementaciones de mala calidad generadas por el multiplexado de la entrada analógica del conversor en diferentes canales analógicos.

La subtarea de muestro debe configurar el multiplexor para seleccionar el canal correcto, indicarle al conversor cuándo debe comenzar a convertir, esperar hasta que la conversión esté realizada y leer el dato convertido. Durante este proceso, la subtareas de muestreo no puede ser apropiada por otra subtarea de

muestreo que requiera la utilización del conversor A/D. Muchos conversores A/D toman un tiempo no despreciable y no constante para convertir la señal analógica, produciendo un muestreo no periódico de la señal de entrada. Esta variación en el periodo es lo que se conoce como jitter.

Las subtareas de control pueden ser ordenadas en cuatro configuraciones principales, las cuales son descriptas en las siguientes subsecciones.

4.2 Caso A. Jitter de entrada y de salida.

La subtarea de muestreo es ejecutada periódicamente. Al finalizar ésta, es ejecutada la subtarea de cálculo y finalmente la subtarea de actuación. Supongamos que tenemos un sistema de tiempo real, como muestra la Figura 4, que consiste en dos tareas: la tarea τ_0, que no es de control y la tarea τ_1, de control constituida por sus tres subtareas correspondientes: subtarea de muestreo, M^i; subtarea de cálculo, C^i y subtarea de actuación, A^i donde i es el índice de invocación. A la tarea τ_0 se le asigna mayor prioridad que a la tarea τ_1.

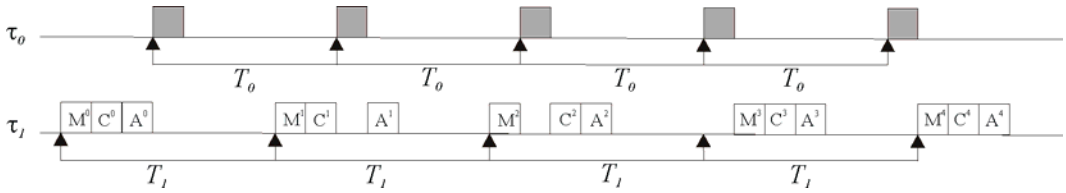

Figura 4. Caso A: Se produce jitter en la tarea de control debido a que la tarea que no es de control tiene mayor prioridad.

En la Figura 4, se puede observar el impacto de la interferencia de tareas de mayor prioridad. En este caso, el arribo de una tarea de mayor prioridad puede conducir a retrasos en: la activación de la tarea, la actuación final o ambos a la vez, resultando en jitter de entrada y de salida.

El jitter de muestreo sobre la tarea τ_1, se produce debido a que el intervalo entre M^2 y M^3 es mayor que T_1, mientras que el intervalo entre M^3 y M^4 es menor que T_1. El jitter de actuación se observa debido a que el intervalo entre A^0 y A^1 es mayor que T_1 mientras que el intervalo entre A^3 y A^4 es menor que T_1.

4.3 Caso B. Solución al jitter de entrada.

El problema anterior de jitter de entrada puede ser minimizado si la tarea se divide en dos subtareas: la subtarea de muestreo y el resto, donde cada parte es ejecutada a una prioridad diferente. Corriendo todas las subtareas de muestreo a la mayor prioridad, el jitter de muestreo se reduce notablemente. La Figura 5 muestra la ejecución de las tres subtareas de la tarea de control τ_1 en dos niveles de prioridad diferentes.

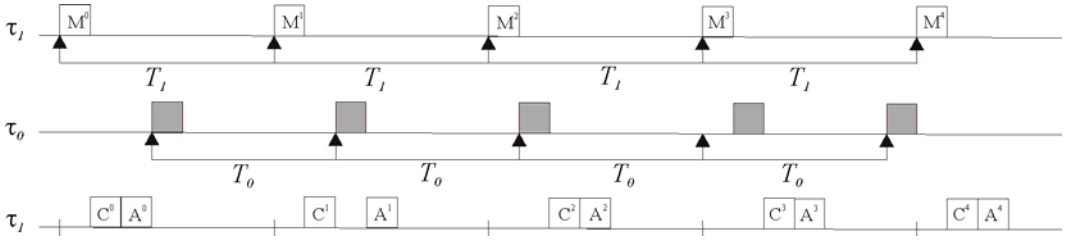

Figura 5. Caso B: A la subtarea de muestreo de la tarea de control 1 se le asigna la mayor prioridad evitando jitter de muestreo.

Debe tenerse en cuenta que ambas subtareas deben tener el mismo período, así nos aseguramos que, aunque la subtarea de calculo-actuación esté lista para la ejecución al mismo tiempo, nunca se ejecutará antes que la de muestreo debido al ordenamiento de prioridad.

4.4 Caso C. Solución al jitter de entrada y de salida, con retardo en el cálculo de control.

La configuración previa soluciona el problema de jitter de entrada pero no el de salida. Se puede realizar una simple extensión de la misma para solucionarlo, corriendo la subtarea de actuación a una prioridad mayor junto con la de muestreo.

En este caso, la subtarea de muestreo es ejecutada periódicamente y la subtarea de actuación justo después de que ésta finaliza. No obstante, la salida resulta del cálculo de la invocación previa. Todas las subtareas de cálculo corren a baja prioridad como antes. Este retardo afecta la eficiencia de control pero puede ser considerado en el modelo de control de tiempo discreto. Debido a que éste es un retardo *fijo* de precisamente T unidades de tiempo, su impacto puede ser

modelado exactamente. La Figura 6 muestra la ejecución de las tres subtareas de control.

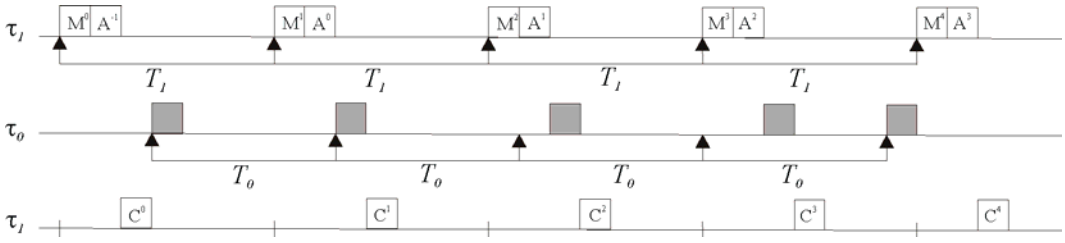

Figura 6. Caso C: A ambas subtareas de muestreo y actuación se le asigna la mayor prioridad para evitar tanto el jitter de entrada y salida. Se introduce retado en la actuación.

Se observa en esta figura que tanto el jitter de entrada como el de salida son eliminados, produciéndose sólo un retardo en la actuación, que puede ser modelado precisamente.

4.5 Caso D. Solución al jitter de entrada y de salida, con menor retardo en el cálculo de control.

El caso anterior es un caso particular del problema más general teniendo las distintas subtareas a diferentes niveles de prioridad. En este caso, la subtarea de muestreo es ejecutada periódicamente y la subtarea de actuación es ejecutada aun con un retardo fijo desde el período de muestreo. El retardo es elegido de tal manera que la subtarea de cálculo siempre finalice antes del comienzo de la de actuación. En este caso no hay jitter de salida y hay un retardo menor que en el caso anterior.

La Figura 7 muestra la ejecución de las tres subtareas de la tarea τ_1 en dos niveles de prioridad diferentes.

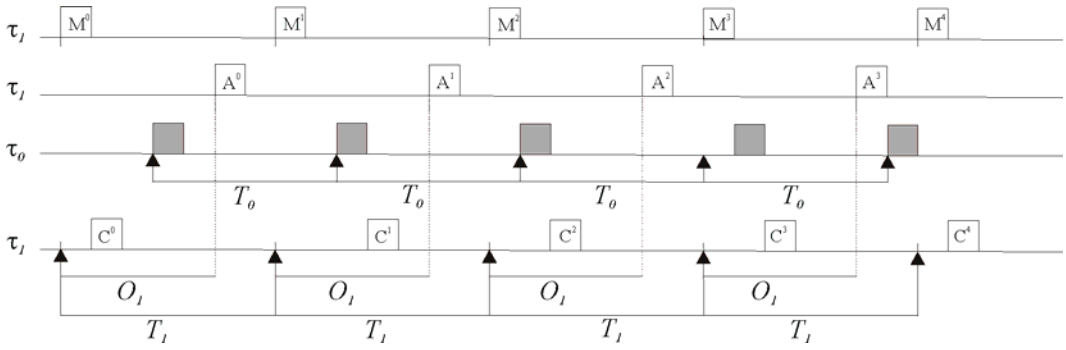

Figura 7. Caso D: la subtarea de actuación es invocada después de un determinado offset. Tanto el jitter de muestreo como el de actuación son eliminados.

La limitación de ésta aproximación es debido a la complejidad en la implementación y a la sobrecarga, ya que requiere muchos recursos del sistema debido a que la separación de la tarea τ_1 en tres subtareas, para el sistema es como si estuviera corriendo tres tareas distintas.

4.6 Conclusiones

En este capítulo se mostraron los distintos modelos de las tareas de control que son propuestos en la literatura de tiempo real. Esto modelos son utilizados para la obtención de mecanismos que pueden mejorar la eficiencia de control debido a las perturbaciones que se introducen en la diagramación de tareas de tiempo real.

En el capítulo siguiente se presentan algunos métodos flexibles de planificación estudiados en la literatura de tiempo real.

Capítulo 5

Métodos Flexibles en Sistemas de Tiempo Real

En la comunidad de tiempo real, ha habido extensas investigaciones en métodos flexibles de planificación a través de diferentes trabajos. No obstante, sólo unos pocos de éstos son realmente aplicables a la teoría de control. En este Capítulo repasamos algunos de estos trabajos.

5.1 Control realimentado elástico

En [11] y [14], se propone el control realimentado elástico. En estos trabajos se diseña un algoritmo adaptivo para cambiar el período de muestreo de la tarea de acuerdo a la carga del sistema de tiempo real. Cuando la carga del sistema se relaja, el período de las tareas es reducido tanto como sea posible. Cuando el sistema es sobrecargado, entonces el período de las tareas es incrementado para reducir la carga. Estos mecanismos muestran una alta eficiencia de tiempo real, pero ellos pueden no ser convenientes para estrategias de control complejas.

En el Experimento 3 que se mostrará en el Capítulo 6, se analizan los efectos que producen el cambio del período de muestreo de la tarea en tiempo de ejecución, sobre la aplicación de control. Posteriormente, en el Capítulo 9, se presenta un mecanismo basado en la idea de estos trabajos, que modifica gradualmente el período de las tareas de control logrando que el error introducido sea acotado, incluso en estrategias de control complejas, alcanzando una alta flexibilidad de tiempo real, al mismo tiempo que mantiene la eficiencia de control.

5.2 Controladores de modelo predictivo y Compensación de jitter

El Controlador de Modelo Predictivo, propuesto en [24] y [15] está basado en la configuración del Caso D desarrollado en la Sección 4.5. La subtarea de muestreo es implementada como una rutina de interrupción por tiempo con un nivel de prioridad alto. La flexibilidad de esta aproximación está en definir la subtarea de computación en forma iterativa en donde mayor tiempo de ejecución produce una computación más precisa. Esta tarea iterativa es ejecutada hasta un cierto tiempo en el cual el retardo de salida es considerado aceptable. La optimización de la computación es mejorada tanto como esta subtarea es ejecutada.

En [33], se propone un mecanismo para compensación del jitter que modifica en tiempo de ejecución el modelo en tiempo discreto del controlador. Las tablas de parámetros de control son almacenadas en memoria para cambiar los del controlador en tiempo de ejecución. Se utilizan técnicas de tiempo real flexibles para la planificación de las tareas.

Aunque ambas propuestas intentan reducir el jitter de subtareas de muestreo, éstos pueden conducir a variaciones de jitter si varias subtareas de muestreo son implementadas. En el Experimento 2 que se mostrará en el Capítulo 6, se analiza el efecto que el jitter de muestreo produce sobre la aplicación de control.

5.3 Restricciones temporales de tareas de control.

En [32], se define el coeficiente de Calidad de Control para parametrizar el comportamiento de un sistema de control. Tal trabajo concluye que un mejor coeficiente de Calidad de Control es alcanzado cuando la tarea de control es invocada a una frecuencia alta (período pequeño).

En dicho trabajo se han realizado simulaciones utilizando el péndulo invertido pero no se han mostrado resultados experimentales. En el Experimento 1 que se mostrará en el Capítulo 6, se establece que el coeficiente de Calidad de Control no representa la eficiencia de la aplicación de control debido a que períodos de muestreo pequeños pueden conducir a comportamientos incontrolados del sistema.

5.4 Planificación de tiempo real débil-duro (weakly-hard).

El modelo skip-over fue introducido por Koren y Sasha ([29]). El algoritmo de planificación skip-over "salta" algunas invocaciones de la tarea de tiempo real de acuerdo al *factor de salto s*. Si una tarea tiene un factor de salto de *s*, una invocación de ésta será salteada de cada *s* invocaciones consecutivas. Los algoritmos de planificación skip-over son on-line y de baja complejidad pero pueden no ser extendidos a sistemas con restricciones temporales arbitrarias. En [12] y [13], los saltos invocaciones son aprovechados para minimizar el tiempo de respuesta de las tareas aperiódicas.

En [21], fue introducido el concepto de vencimientos (m, k)-firm para modelar tareas que tienen que alcanzar m vencimientos cada k invocaciones consecutivas en el contexto de paquetes de redes de comunicación.

En [9] y [5], es introducida la noción de restricciones de tiempo real weakly-hard como una generalización del concepto de (m, k)-firm para cubrir otros tipos de modelos de pérdidas de invocación. En [10], es propuesto un mecanismo de planificación basado en dual priority para reforzar el modelo de tiempo real weakly. Se proveen condiciones de planificabilidad suficientes para garantizar que cada restricción de tiempo real weakly será satisfecha en tiempo de ejecución.

En el Experimento 4 que se mostrará en el Capítulo 6, se analizan las perturbaciones que las pérdidas de vencimientos producen sobre la aplicación de control, para conocer si ésta puede tolerar ocasionales pérdidas de vencimientos.

5.5 Conclusiones

En éste capítulo se introdujeron varios mecanismos flexibles de planificación que han sido investigados para implementarlos en sistemas de control.

En el capítulo siguiente se analizan los efectos que producen cada uno de estos mecanismos sobe la aplicación de control realizando simulaciones con la herramienta TrueTime basada en Matlab/Simulink que tiene en cuenta todas las características reales de la aplicación.

Capítulo 6

Análisis de Perturbaciones

En esta sección se analizan los efectos que los sistemas de tiempo real producen en aplicaciones de control. Las conclusiones obtenidas al final de esta sección serán utilizadas en los capítulos 7 y 9 para proponer diseños de mecanismos de planificación que solucionen o minimicen dichos efectos adversos. Se evalúan todas las posibles interferencias que son introducidas cuando las aplicaciones son implementadas en la realidad.

6.1 Introducción

A través de un caso de estudio real, se investiga y cuantifica el impacto que producen las discrepancias entre las hipótesis del modelo generalmente usado en sistemas de tiempo real y el modelo usado en la teoría de control. En particular se analizan los casos introducidos en el Capítulo 4 los cuales los recordamos aquí:

- La selección de período de muestreo correcto y el impacto que produce el cambio de éste en las tareas de control.
- El impacto que producen los jitters de entrada y salida.
- Los modelos de implementación de tareas de control

- El impacto que produce la pérdida de invocaciones de tareas.

Se muestra que algunos de los requerimientos de control no pueden ser expresados en un modelo de proceso tradicional de tiempo real duro y que algunas de las perturbaciones producidas por un planificador clásico de tiempo real duro pueden tornar la aplicación incontrolable, mientras que un planificador flexible, no. Además, se concluye que un buen diseño de una aplicación debe tener en cuenta tanto la mecánica, la dinámica y las propiedades de tiempo real de la aplicación.

El caso estudiado en este Capítulo está basado en el clásico péndulo invertido. Éste es usado ampliamente en control debido a que ilustra ideas en control lineal tales como estabilización de sistemas inestables ([3], [39]). Además los resultados obtenidos con este sistema son extensibles a otras áreas de aplicación como la mecánica, la termodinámica y la mecatrónica, entre otras. Para comparar los resultados experimentales con las simulaciones, se obtuvo un modelo analítico de nuestro péndulo real y usamos TrueTime ([25]) para realizar las simulaciones.

6.2 Caso de Estudio: El Péndulo Invertido.

Para mostrar las influencias que un sistema de tiempo real produce sobre las aplicaciones de control, analizamos como caso de estudio el clásico sistema de control de péndulo invertido.

El péndulo utilizado consiste en un brazo de 50 cm de longitud y masa de 20.67 g. En un extremo está unida a una masa extra de 50.65 g y el otro extremo al eje de un motor de corriente continua que tiene un torque de $4.369 \cdot 10^{-6}$ N. El rango de voltaje aplicado al motor es de -5 Volts a +5 Volts. El torque del motor es proporcional al voltaje. Cuando el voltaje es negativo, el torque es contrario a las agujas del reloj, mientras que es en sentido de las agujas del reloj cuando el voltaje es positivo.

Una estrategia de control toma el brazo y lo mantiene en la posición vertical cuando éste entra a la zona angular entre -0.14 radianes y +0.14 radianes ($\approx \pm 8^o$) desde la posición vertical. El *controlador vertical* cambia el voltaje del motor, de acuerdo a:

$$y = -35.5 \cdot \alpha - 7.8 \cdot \frac{d\alpha}{dt} \qquad (14)$$

donde α es el ángulo desde la posición vertical, y $d\alpha/dt$ es la velocidad angular.

Como el torque del motor no es suficiente para llevar directamente el brazo desde la posición de reposo hasta la posición vertical, una estrategia de control de ascenso debe balancear el brazo para aumentar su energía en cada oscilación.

El *controlador de ascenso* cambia el voltaje del motor de acuerdo a:

$$y = -70 \cdot \alpha - 13 \frac{d\alpha}{dt} \qquad (15)$$

Los coeficientes de la estrategia de control fueron elegidos de acuerdo a características particulares del péndulo aplicando criterios de control. En la Figura 8 vemos un esquema del péndulo invertido donde se pueden apreciar las zonas del mismo.

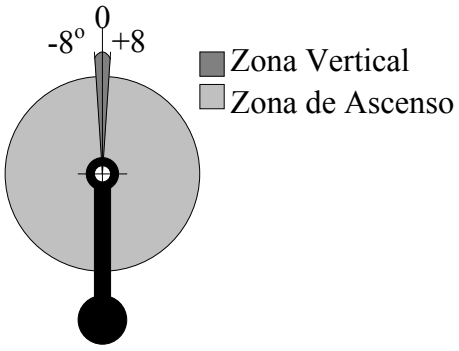

Figura 8. Zonas del péndulo

Como el rango dinámico del voltaje del motor es limitado, el voltaje dado por las estrategias de control será saturado entre –5 Volts y +5 Volts.

El ángulo del brazo es medido por un encoder a ranuras con resolución de $\pi/2200$ radianes. La velocidad es obtenida indirectamente dividiendo la diferencia entre el muestreo de dos ángulos consecutivos por el período de muestreo (T). De esta manera, la forma en tiempo discreto de la Ecuación 15 es:

$$y[k] = \frac{-70 \cdot \alpha[k] - 13 \cdot (\alpha[k] - \alpha[k-1])}{T} \qquad (16)$$

Ésta puede ser expresada en forma de modelo de espacio de estados de la siguiente manera:

$$x_1[k+1] = \alpha[k]$$
$$y[k] = \left[\frac{13}{T}\right] \cdot x_1[k] + \left[-70 - \frac{13}{T}\right] \cdot \alpha[k] \qquad (17)$$

La Ecuación 14 puede ser expresada en espacio de estados en tiempo discreto de la misma manera.

Como los encoders tienen una resolución finita, se introduce un error en la lectura del ángulo y por consiguiente en el cálculo de la velocidad angular. En consecuencia, se produce un error en la salida del controlador debido a que tanto el ángulo como la velocidad angular son medidas con error.

En los siguientes experimentos evaluamos, usando TrueTime, herramienta basada en Matlab (ver Anexo B para más información), el impacto de las características descriptas anteriormente, las cuales son llamadas: impacto del jitter, cambio de período de muestreo en tiempo de ejecución, y pérdida/salto de invocaciones de las tareas de control. No implementamos directamente los mecanismos flexibles, pero emulamos en forma independiente y controlada las perturbaciones que esos mecanismos producen sobre el control del péndulo invertido con el objetivo de obtener un mejor análisis de las causa/efecto. La emulación es realizada introduciendo en el sistema, jitter, offset, salto de invocaciones y variaciones del período en la ejecución de la tarea de control.

El kernel de TrueTime simula la ejecución de la estrategia de control de ascenso del sistema hasta que el brazo del péndulo alcanza la posición vertical. La evaluación de la eficiencia es realizada considerando el tiempo de establecimiento del péndulo en la posición vertical, debido a que la estrategia de control vertical es muy sensible al jitter. Debido a que el torque del motor no es suficiente para elevar directamente el brazo del péndulo, la estrategia de ascenso balancea el brazo del péndulo para aumentar la energía del mismo en cada oscilación ([3]). La energía debe ser transferida de acuerdo a la oscilación natural del péndulo, de lo contrario, la energía no será acumulada apropiadamente y el resultado puede ser indeseable (ruido, vibración, calor). Cuando la estrategia de control de ascenso es

afectada por jitter, la energía puede no ser transferida adecuadamente al péndulo. Por otro lado, la estrategia de ascenso debe transferir el control a la estrategia de control vertical cuando el brazo se encuentre en la zona de control vertical con velocidad angular cero. De otra manera, puede ocurrir que la estrategia de control vertical no pueda frenar la inercia del brazo y el péndulo de una vuelta completa.

Cabe destacar que en las experiencias se muestran los puntos en los cuales el tiempo de establecimiento es menor o igual a 120 seg. Los puntos en los cuales el tiempo de establecimiento es mayor que este valor, fueron considerados como que el péndulo oscila y nunca alcanza la posición vertical.

La Figura 9 muestra el esquema implementado con TrueTime.

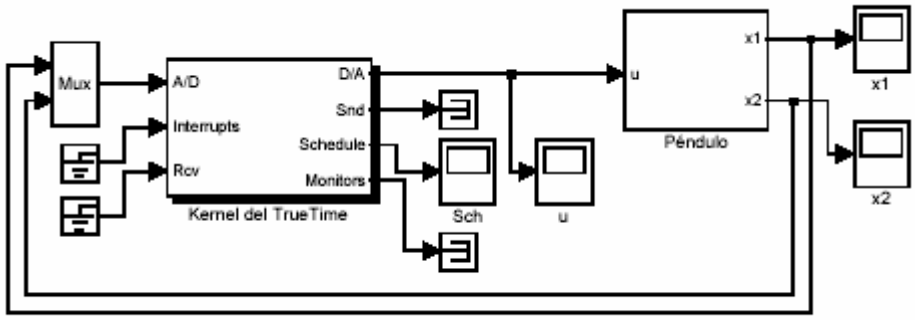

Figura 9. Sistema implementado con TrueTime

En la figura se puede notar cómo el kernel de TrueTime envía la acción de control discreta al sistema a controlar a través del conversor digital-analógico (D/A). Las salidas del sistema son realimentadas al kernel de TrueTime a través del conversor analógico-digital (A/D) para calcular la siguiente acción de control. Finalmente, el kernel tiene una salida más que es el *schedule* que entrega la planificación producida por el algoritmo de planificación usado.

6.3 Experiencias realizadas.

En las sub-secciones siguientes, presentamos cuatro experimentos para mostrar las influencias que los sistemas de tiempo real producen sobre las aplicaciones de control.

6.3.1 Experimento 1. Selección del período de muestreo de la tarea de control.

En este experimento, analizamos el desempeño de la tarea de control de acuerdo a su período. Medimos el tiempo de establecimiento para la misma tarea considerando diferentes períodos en un rango desde 10 mseg a 400 mseg. No hubo interferencias de otras tareas del sistema y consecuentemente el jitter es igual a cero. Se consideraron siete diferentes tiempos de ejecución para la tarea de control: 0 mseg (computación instantánea), 0,5 mseg, 1 mseg, 1,3 mseg, 1,5 mseg, 2 mseg y 3 mseg. La Figura 10 muestra el desempeño de la tarea de control de acuerdo a su período de muestreo.

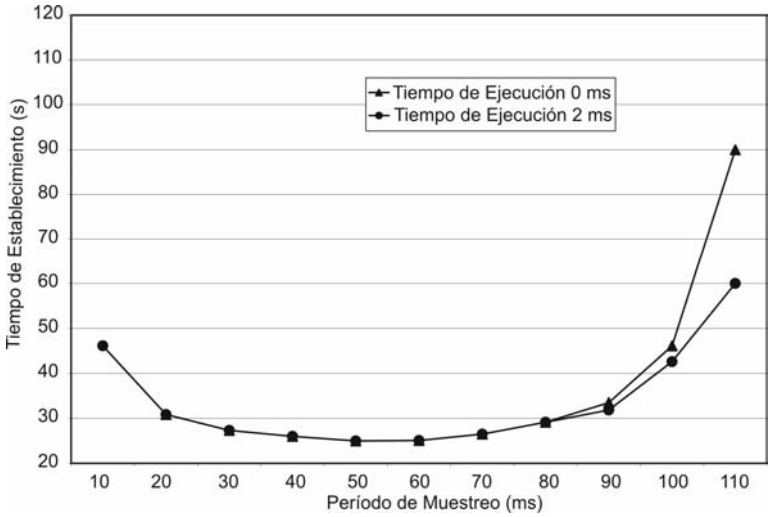

Figura 10. Desempeño de la tarea de control de acuerdo a su período de muestreo

De la Figura 10 podemos concluir que el rango de períodos de muestreo óptimo es de 20 mseg a 80 mseg. Pequeños períodos de muestreo producen un gran error en el cálculo de la velocidad y consecuentemente largos tiempos de establecimiento. Grandes períodos de muestreo reducen el ancho de banda de la estrategia de control y consecuentemente no puede reaccionar según la dinámica de la aplicación.

El tiempo de ejecución de la tarea de control tiene una gran influencia cuando el período está fuera del rango óptimo. Este es un efecto compuesto producido por el error introducido en el cálculo de la estrategia de control y la actualización de la salida. Ambas causas pueden ser aditivas o substractivas de acuerdo a la dinámica particular de la aplicación. En este caso particular, un retardo grande en la

actualización de la salida (es decir un tiempo de ejecución grande) mejora el tiempo de establecimiento del péndulo. Sin embargo, los experimentos realizados en los péndulos con diferentes características producen un efecto opuesto.

6.3.2 Experimento 2. Análisis de las perturbaciones del jitter.

En este experimento, analizamos el efecto que produce el jitter sobre la eficiencia de control. El jitter fue producido por el ingreso al sistema de una tarea de mayor prioridad, que no es de control, que comparte el procesador con la tarea de control. El período de la tarea que no es de control, fue elegido igual al doble del de la tarea de tiempo real. De esta manera, podemos modificar el jitter introducido a la tarea de tiempo real modificando el tiempo de ejecución de la tarea que no es de control.

Se realizó el mismo experimento que el anterior para diferentes tiempos de ejecución de la tarea de no-control iguales a: 1 mseg, 2 mseg, 3 mseg, 4 mseg, 5 mseg, 8 mseg y 10 mseg. En la Figura 11, mostramos sólo el efecto de 1 mseg, 5 mseg y 10 mseg de jitter sobre la eficiencia de una tarea de control con tiempo de ejecución igual a 1,5 mseg. Para tareas de control con otros tiempos de ejecución (0 mseg, 0,5 mseg 1 mseg, 1,3 mseg, 2 mseg, y 3 mseg), los resultados obtenidos fueron similares.

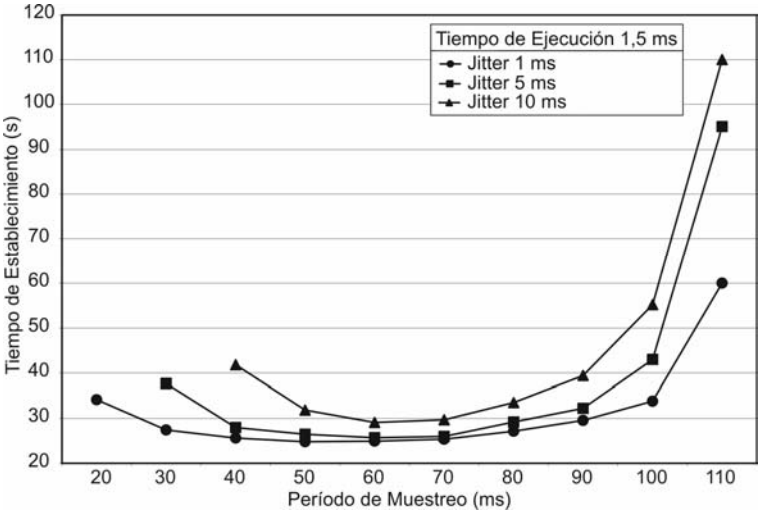

Figura 11. Efecto del jitter sobre la eficiencia de control.

En la Figura 11 podemos notar que las aplicaciones de control son muy sensibles al jitter. Un jitter mayor que el 20% del período de una tarea de control,

puede tornar la aplicación inestable o incontrolada. Es decir, el péndulo puede tardar un tiempo muy alto en alcanzar su posición vertical o puede oscilar por tiempo indeterminado. Para un adecuado desempeño de muchas aplicaciones de control, es recomendable que el jitter de una tarea de control se mantenga por debajo de un 10% del período de muestreo.

6.3.3 Experimento 3. Variación el período de la tarea en tiempo de ejecución.

En este experimento analizamos el efecto que produce la ejecución de una tarea de control con un periodo diferente sobre la eficiencia de control. Ejecutamos la tarea de control con un período diferente del considerado en el programa desarrollado para la tarea. Se han realizado varias experiencias para diferentes períodos de muestreo. En la Figura 12, se muestra una experiencia donde la tarea de control es programada para ser ejecutada a un período de muestreo igual a 50 mseg pero se ejecuta con períodos iguales a 52,5 mseg, 55 mseg, 57,5 mseg, 60 mseg y 62,5 mseg, los cuales corresponden a errores iguales al 5%, 10%, 15%, 20% y 25% respectivamente. Se eligió el período base de 50 mseg debido a que está dentro del rango óptimo encontrado en el experimento 1.

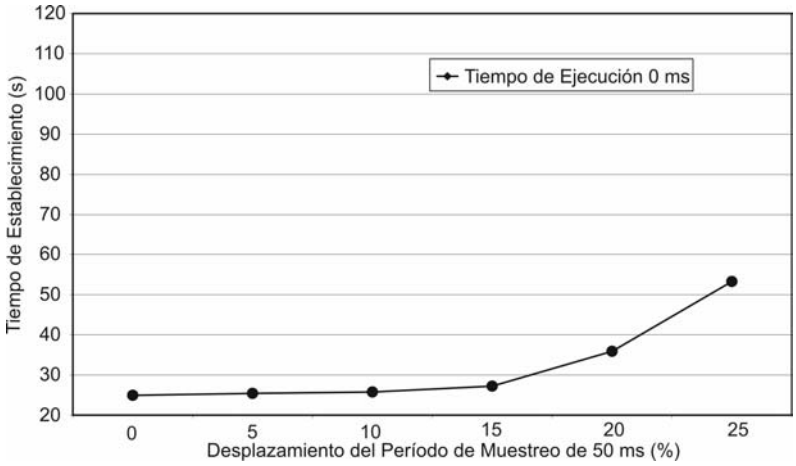

Figura 12. Efecto que un desplazamiento del período de una tarea presenta sobre la eficiencia de control en tiempo de ejecución.

De la Figura 12 podemos concluir que el sistema tolera desplazamientos de hasta un 15% sin gran impacto sobre el tiempo de establecimiento cuando el período de muestreo es igual a 50 mseg. La tarea de control se vuelve más sensible a desplazamientos en las invocaciones cuando el período es mayor. Por

otro lado, no se obtuvieron adecuados tiempos de establecimiento cuando los desplazamientos fueron mayores a un 30%. Mayores períodos son más estrictos acerca de los desplazamientos que puede soportar, es decir que cuanto más aumentamos el período de muestreo, menos desplazamiento del mismo podemos realizar para obtener una buena eficiencia de control.

6.3.4 Experimento 4: Salteando invocaciones de la tarea.

En este experimento, se saltea una invocación de cada *s* consecutiva para mostrar el efecto que produce sobre la eficiencia de control la pérdida de invocaciones. Se consideró un tiempo de ejecución instantáneo para la tarea de control. En la Figura 13, mostramos el efecto de perder una invocación de cada *s* = 5 y *s* = 4.

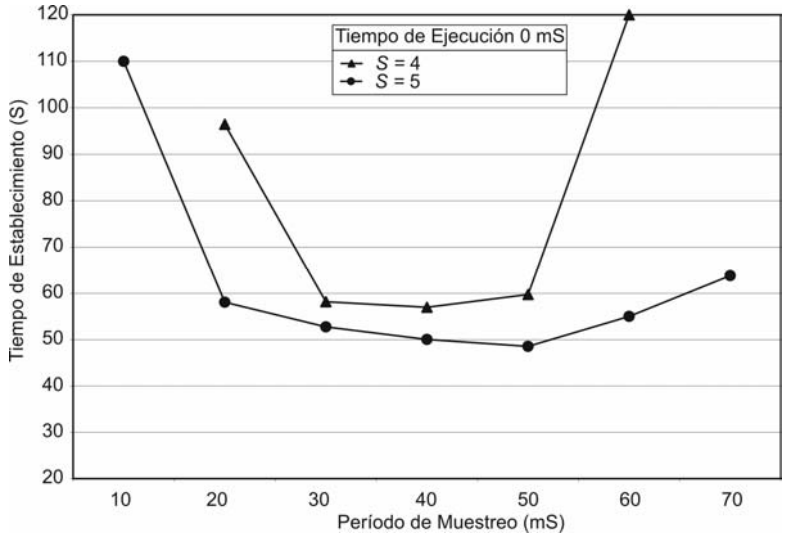

Figura 13. Efecto que produce la pérdida de invocaciones sobre la aplicación de control.

En la Figura 13 podemos notar que el sistema soporta peder una de cada 5 o más invocaciones cuando se elige para muestrear un período dentro del intervalo de muestreo óptimo, pero el tiempo de establecimiento crece ligeramente comparando con el Experimento 1, en el cual no teníamos pérdida de invocaciones. El efecto de saltearse una invocación es menos notable para períodos de muestreo menores. Por otro lado, para intervalos de muestreo fuera del intervalo óptimo, no se pueden saltear una en menos de 7 invocaciones

6.4 Análisis de resultados.

Hay un rango óptimo para el período de muestreo de las tareas de control. Eligiendo el período fuera de este rango, la aplicación se torna muy sensible al tiempo de ejecución de la tarea y el tiempo de establecimiento que se obtiene se aleja del óptimo. Desde el punto de vista de tiempo real, ejecutando la tarea de control más frecuentemente no se mejora la calidad de control (oponiéndose a lo expuesto en [33]). El rango de muestreo óptimo debe ser tenido en cuenta cuando se analiza la planificabilidad del sistema de tiempo real.

Mostramos que las hipótesis ampliamente conocidas en la literatura de tiempo real que consideran vencimientos iguales a períodos, no son válidas cuando tratamos con tareas de control. Debe remarcarse que la influencia del jitter es concerniente al período de la tarea y por lo tanto podemos elegir un período más grande para una planificación más relajada desde el punto de vista de tiempo real.

El experimento 3 muestra que la aplicación de control puede soportar un desplazamiento del período de muestreo en tiempo de ejecución de hasta un 15%.

El análisis de tiempo real es basado en la hipótesis que los sistemas de tiempo real no pueden perder ningún vencimiento o un número obligado de vencimientos. Sin embargo, las aplicaciones de control pueden soportar algunas pérdidas de vencimientos. Existe un rango de períodos de muestreo, (en nuestro caso de estudio entre 30 mseg y 50 meg) en el cual el sistema es más robusto a los saltos de invocaciones ([21]).

6.5 Conclusiones.

En este Capítulo se analizó los efectos que los sistemas de tiempo real producen sobre las aplicaciones de control. La teoría clásica de control digital considera el muestreo de entradas y la actualización de las salidas estrictamente periódicas. Por otra parte, los sistemas de tiempo real duro consideran tareas periódicas pero pueden presentarse diversas perturbaciones de control debido a las políticas de planificación. Estas perturbaciones de tiempo real pueden tornar inestable e incontrolable a la aplicación de control.

Varios criterios de planificación flexibles de tiempo real han sido desarrollados para mejorar diferentes características de tiempo real en tiempo de ejecución, pero no han expuesto resultados sobre las repercusiones que éstos tienen sobre la aplicación de control en sí misma.

Muchas veces la carencia de precisión en los sensores y actuadores del sistema, no justifican la mejora en el desempeño del sistema en tiempo real. Tanto el sistema de tiempo real como la aplicación de control deben ser analizados en forma conjunta para lograr el mejor diseño.

Por medio de simulaciones se emularon las perturbaciones que la planificación flexible de tiempo real puede producir sobre la aplicación de control. Se mostró que algunos de los requerimientos de control no pueden ser expresados en un modelo de proceso tradicional de tiempo real duro. Además, se mostró que algunas de las perturbaciones producidas por un planificador clásico de tiempo real duro, pueden tornar incontrolable a la aplicación, mientras que un esquema flexible es capaz de producir un comportamiento controlable.

Por lo tanto, es conveniente analizar la factibilidad de técnicas de tiempo real flexibles, cuando se requiera la implementación de aplicaciones de control, tales como las que se desarrollan en los a Capítulos 7 y 9.

Capítulo 7

Planificador de Bajo Jitter. Un Mecanismo Flexible de Planificación de Tareas de Control de Tiempo Real

En esta sección proponemos un mecanismo de planificación basado en el concepto de "Dual Priority" ([17]) para reducir los efectos adversos que el jitter produce en sistemas de control automático. Por otro lado, con el uso de un esquema Dual-Priority, el tiempo de respuesta de las tareas que nos son de control en sistemas de tiempo real es mejorado. El mecanismo propuesto promueve las tareas de control del sistema de acuerdo al nivel de sensibilidad al jitter que tenga el sistema en un determinado momento. Sobre el final del capítulo mostramos que el mecanismo propuesto mejora tanto la eficiencia de tareas de control como el tiempo de respuesta de las tareas que no lo son.

7.1 Introducción

En el caso ideal de una aplicación de control, se asume que el sensado, la computación y la actuación no consumen tiempo y por lo tanto la tarea de control es ejecutada exactamente a kT instantes de tiempo.

No obstante, el comportamiento real de aplicaciones de control, puede diferir del ideal debido a cuestiones prácticas de la implementación tales como exactitud del sensor, sensibilidad y rango de los actuadores, truncamiento de entradas y salidas, precisión mecánica y de cómputo, condiciones ambientales, entre otras. Las estrategias del control se deben diseñar tan robustas como sea posible para evitar estos efectos secundarios.

Varios trabajos han analizado los efectos que producen la planificación de tareas en tiempo real sobre aplicaciones de control. Entre los más importantes podemos citar los siguientes: [2], [16], [33], [4], [23], [47] y [42]. Todos estos trabajos proponen diferentes modelos de tareas de control para reducir el jitter pero ninguno de ellos trata además de mejorar el tiempo de respuesta de las tareas que no son de control.

En este Capítulo se propone un mecanismo de prioridades dinámicas que reduce los efectos adversos producidos por el jitter en sistemas de control. Debido a que el jitter es variable durante el tiempo de funcionamiento del sistema, el mecanismo propuesto, al que llamamos “Planificador de Bajo Jitter”, de ahora en adelante PBJ, reacciona a los cambios en las condiciones de carga de acuerdo a cuál sensible es al jitter la aplicación de control. Analizamos el rango de los sensores de entrada y actuadores de salida para detectar cuándo una tarea es más sensible al jitter. También examinamos las condiciones en las cuales la aplicación de control necesita una precisión más estricta de sus parámetros temporales. Por ejemplo, si una aplicación se encuentra en estado transitorio, ésta puede ser más sensible al jitter que una en estado estacionario. En la Sección 7.2 se describe cómo detectar cuando las tareas son menos sensibles al jitter. De este modo, podemos asignarle el procesador a la tarea más sensible al jitter. El desempeño de PBJ es comparado con los planificadores de prioridades fijas (FP) y de prioridades dinámicas (EDF). El caso estudiado es basado en el clásico péndulo invertido. Se

obtuvo un modelo analítico de nuestro péndulo real y usamos TrueTime ([25]) para comparar la eficiencia de nuestro mecanismo con la que se obtiene utilizando Prioridades Fijas (FP) y Menor Tiempo al Vencimiento (EDF) ([30]).

7.1.1 Planificador Dual- Priority

El planificador Dual-Priority fue propuesto en [17] para mejorar el tiempo de respuesta de tareas que no son de tiempo real en sistemas de tiempo real. Se basa en tres bandas de prioridad: Superior, Media e Inferior. Las tareas asignadas a una misma banda son planificadas usando una disciplina de prioridad determinada.

Cuando no hay tareas listas para ser ejecutadas en la banda Superior, la tarea de mayor prioridad lista para ser ejecutada en la banda Media, es ejecutada. Si tampoco hay alguna tarea lista para la ejecución en esta banda, entonces la tarea de mayor prioridad lista para la ejecución en la banda Inferior será ejecutada.

En un mecanismo dual-priority, una tarea puede ser promovida desde su banda a una banda de mayor prioridad en tiempo de ejecución para cumplir con sus requerimientos de tiempo real.

Davis et. al. propuso el siguiente criterio de asignación de prioridades: tareas que no son de tiempo real son asignadas a la banda Media mientras que las tareas de tiempo real son asignadas inicialmente a la banda Inferior y cuando existe riesgo de pérdida de vencimientos, son promovidas a la banda Superior.

Las tareas en la banda Superior son planificadas bajo una disciplina de prioridades fijas, mientras que tanto en la banda Media como en la Baja, la disciplina usada es elegida para mejorar la eficiencia de algún criterio de comportamiento.

7.2 Planificador de Bajo Jitter: PBJ

En esta sección describimos el mecanismo de planificación propuesto para mejorar tanto el desempeño de tareas de control como el tiempo de respuesta de tareas que no son de control en un sistema de tiempo real.

Una tarea de control realiza una cierta función: implementar una estrategia de control para obtener el comportamiento controlado deseado en la aplicación.

Conforme a un cambio en las entradas del controlador éste calcula la función de control para actualizar las salidas al actuador.

La estrategia de control debe ser ejecutada periódicamente. Cuando un sistema está en estado estacionario, los cambios en las entradas son casi despreciables y las salidas se mantienen en el mismo valor. En este caso, reducir el jitter de la tarea de control no producirá ningún efecto sobre el desempeño de control. De otro modo, es decir si el sistema se encuentra en estado transitorio, cuando las entradas del sistema cambian, las salidas deben ser aplicadas tan pronto como sea posible de acuerdo a las estrictas limitaciones de tiempo que las técnicas de modelado de control lo especifican.

Las estrategias de control son generalmente funciones lineales cuyas salidas son aplicadas a actuadores con rango dinámico finito. Por lo tanto, podemos asegurar que no tiene objeto reducir el jitter de una tarea de control cuya salida está sobresaturada.

Teniendo en cuenta la precisión, sensibilidad e histéresis tanto de sensores como de actuadores, no tiene sentido reducir el jitter de las tareas de control si su desempeño no es mejorado en una magnitud equivalente. Por ejemplo, si la precisión del actuador es +/- 0,05V entonces no tiene sentido tratar de incrementar la precisión de temporizado del sistema para obtener una precisión mayor a 0,0005V.

PBJ es un mecanismo de planificación dinámico basado en el esquema dual-priority ([17]) que cambia la prioridad de las tareas de control de acuerdo a su sensibilidad al jitter. Las tareas de control son asignadas a la banda inferior pero en determinado momento pueden promover a la superior. En la banda inferior éstas son planificadas usando Prioridades Fijas. Cuando el planificador detecta que una tarea de control es sensible al jitter, entonces éste la promueve desde la banda inferior a la superior. Cuando deja de ser sensible al jitter, entonces es asignada a la banda inferior nuevamente. Las tareas de seguridad son asignadas siempre a la banda superior para mantener un comportamiento seguro de la aplicación. La banda media del planificador se dedica para las tareas que no son de control para maximizar su tiempo de respuesta, según lo desarrollado por Davis en ([17]).

El criterio que aplica el planificador para promover la prioridad de las tareas de control es:

- La variación de la entrada en la última invocación: Se define un umbral para la variación de la entrada entre dos invocaciones consecutivas para determinar si la tarea de control es o no sensible al jitter. Si la entrada varía en una cantidad despreciable respecto a la precisión del sensor, la tarea de control no es sensible al jitter y por lo tanto no será promovida.
- La variación de la salida en la última invocación: Se define un umbral para la variación de la salida entre dos invocaciones consecutivas para determinar si la tarea de control es o no sensible al jitter. Si la salida de la estrategia de control varía en una cantidad despreciable con respecto a la precisión del actuador, entonces no es sensible al jitter y por lo tanto no será promovida.
- El valor tanto de las entradas como de las salidas en la última invocación: Cuando los valores de las entradas o salidas están fuera del rango de los sensores o actuadores, entonces los efectos del jitter no serán notables. Por ejemplo, la Figura 14 muestra la salida calculada por la estrategia de control y el voltaje aplicado al actuador. Puede notarse que cuando la estrategia de control produce salidas fuera del rango del actuador, la tarea no será afectada por el jitter a causa de que la salida se mantiene en el mismo valor. En este caso, decimos que la tarea no es sensible al jitter y por lo tanto no será promovida. Cabe aclarar que la teoría de control robusto permite diseños que consideran truncamientos de las salidas en estrategias de control ([43])

En los demás casos, consideramos que la tarea de control es sensible al jitter y por lo tanto es promovida.

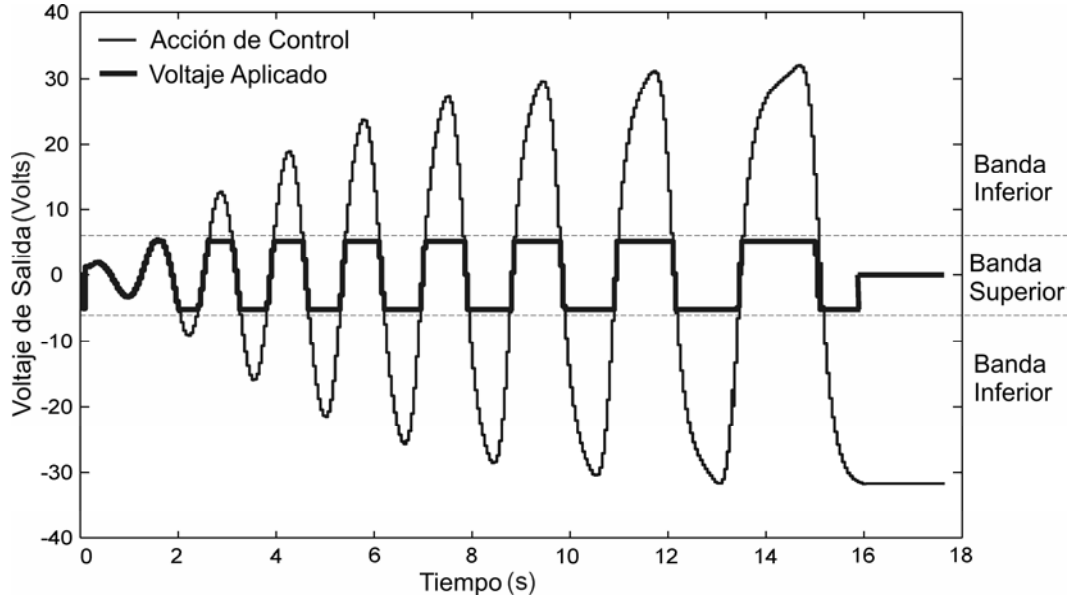

Figura 14. Promoción de prioridad de acuerdo a la salida calculada por la estrategia de control y el voltaje aplicado al actuador

Observamos que el planificador usa el dato obtenido en la invocación anterior de la tarea de control para fijar la prioridad de la siguiente invocación. Por consiguiente, la reacción del planificador se puede retardar a lo sumo un intervalo de muestreo. No obstante, debido a que la estrategia de control es ejecutada con una frecuencia mayor al ancho de banda de la aplicación de control, este retado constante puede ser modelado o despreciado ([43]).

Un análisis más estricto de la robustez de control requiere un conocimiento profundo de la sensibilidad del controlador a la variación de sus parámetros.

Desde el modelo de espacio de estados podemos expresar la ecuación de espacio de estados tanto de la entrada como de la salida del controlador considerando parámetros tales como temperatura, humedad, viscosidad, ruido, fricción, entre otros, como:

$$\begin{aligned} u[k] &= f(x[k], p_1, p_2, \ldots, p_n) \\ y[k] &= g(x[k], p_1, p_2, \ldots, p_n) \end{aligned} \qquad (\mathbf{18})$$

donde $p_1, p_2, ..., p_n$ son los parámetros que integran el espacio de estados.

La sensibilidad de estas funciones respecto a cada parámetro es dada por:

$$S_{u_k}^{p_1} = \frac{\partial u[k]}{\partial p_1} \cdot \frac{p_1}{u[k]}, \ldots, S_{u_k}^{p_n} = \frac{\partial u[k]}{\partial p_n} \cdot \frac{p_n}{u[k]}$$
$$S_{y_k}^{p_1} = \frac{\partial y[k]}{\partial p_1} \cdot \frac{p_1}{y[k]}, \ldots, S_{y_k}^{p_n} = \frac{\partial y[k]}{\partial p_n} \cdot \frac{p_n}{y[k]} \quad (19)$$

Por otro lado, podemos expresar:

- $\Delta u = u_k - u_{k-1}$ como la variación de la entrada en la última invocación.
- $\Delta y = y_k - y_{k-1}$ como la variación de la salida en la última invocación.

Con estas expresiones, definimos:

- $\widehat{S}_u = \max_{j=1\ldots n}\left\{S_{u_k}^{p_j}\right\}$ la máxima sensibilidad del sensor. Esta sensibilidad permitirá especificar un umbral en la variación de la entrada (Δu) que determine que la tarea no es sensible a una variación menor que dicho umbral en la entrada (u).
- $\widehat{S}_y = \max_{j=1\ldots n}\left\{S_{y_k}^{p_j}\right\}$ la máxima sensibilidad del actuador. Esta sensibilidad permitirá especificar un umbral en la variación de la entrada (Δy) que determine que la tarea no es sensible a una variación menor que dicho umbral en la entrada (y).

La información sobre los parámetros de sensores y actuadores se pueden obtener de las hojas de datos que brindan los fabricantes. Dos de los más importantes son la precisión y el rango de operación, que denominados ε_i y ε_o para la precisión del sensor y del actuador respectivamente, y η_i y η_o para el rango de operación del sensor y del actuador respectivamente.

El planificador no promueve a la banda superior de prioridad una tarea de control si la variación de la entrada o la salida no son comparables con la precisión y el rango de operación de los sensores y actuadores. Por consiguiente, la tarea de control será promovida a la banda superior de prioridad si al menos una de las siguientes inecuaciones es falsa:

$$\Delta u \le \lambda_i \cdot \varepsilon_i \cdot \widehat{S}_u, \quad \Delta y \le \lambda_o \cdot \varepsilon_o \cdot \widehat{S}_y$$
$$-\eta_i \le \Delta u \le \eta_i, \quad -\eta_o \le \Delta y \le \eta_o \quad (20)$$

donde λ_i y λ_o son constantes de proporcionalidad.

Podemos concluir que si todas las inecuaciones anteriores se satisfacen, la tarea de control no será promovida a la banda superior. De otro modo, sí será promovida o, si ya se estaba ejecutando en dicha banda, será mantenida.

7.3 Un caso de estudio. El Péndulo Invertido

Como caso de estudio para mostrar las ventajas de PBJ respecto a FP y EDF, usamos el péndulo invertido que ya fue introducido en la Sección 6.2. La idea en este caso es que el procesador deba ejecutar cinco tareas de control (cinco péndulos con distintas características) y cinco tareas que no son de control que introducirán jitter.

Por lo tanto, considerando que el desarrollo de la estrategia de control PD realizada en la Sección 6.2 corresponde al péndulo 1, cabe aclarar que para los otros cuatro péndulos el desarrollo para la obtención de las estrategias de control PD, es de manera similar.

Entonces, las expresiones de las estrategias de control PD para cada uno de los cinco péndulos es la siguiente:

$$\begin{aligned} y_1[k] &= -55 \cdot x_1[k] - 11 \cdot (x_1[k] - x_1[k-1]) / T_1 \\ y_2[k] &= -58 \cdot x_3[k] - 11 \cdot (x_3[k] - x_3[k-1]) / T_2 \\ y_3[k] &= -61 \cdot x_5[k] - 12 \cdot (x_5[k] - x_5[k-1]) / T_3 \\ y_4[k] &= -65 \cdot x_7[k] - 13 \cdot (x_7[k] - x_7[k-1]) / T_4 \\ y_5[k] &= -70 \cdot x_9[k] - 13 \cdot (x_9[k] - x_9[k-1]) / T_5 \end{aligned} \qquad (\mathbf{21})$$

Los coeficientes de los controladores PD fueron diseñados aplicando el método del lugar de las raíces de tal manera que el tiempo de establecimiento es minimizado.

Estas estrategias de control fueron implementadas en la Sección 7.4 en Simulink, usando TrueTime (para obtener más información acerca de esta herramienta, ver Apéndice B) para evaluar el comportamiento de los péndulos bajo diferentes condiciones de planificación en tiempo real.

7.4 Experiencias realizadas

En esta sección evaluamos el desempeño del planificador PBJ. Para realizar las experiencias, se generó de manera aleatoria un conjunto de sistemas de tiempo real. Cada uno de ellos consiste de cinco tareas de control (en este caso las tareas de control son las estrategias de control desarrolladas en la sección previa, para controlar los cinco péndulos), y cinco tareas que no son de control. Se generaron diez sistemas para cada factor de utilización dentro de un rango desde 0,1 a 1 con paso de 0,1. El período de las tareas de control fue acotado entre 50ms y 80ms, ya que como se estudió en el Capítulo 6, es en este rango donde se obtiene el mejor desempeño de control. El de las tareas que no son de control fue acotado entre 5ms y 200ms para obtener valores de jitter variados.

Simulamos cada uno de estos sistemas utilizando TrueTime teniendo en cuenta todos los efectos prácticos de una implementación real. En la Figura **15** mostramos el esquema implementado con TrueTime.

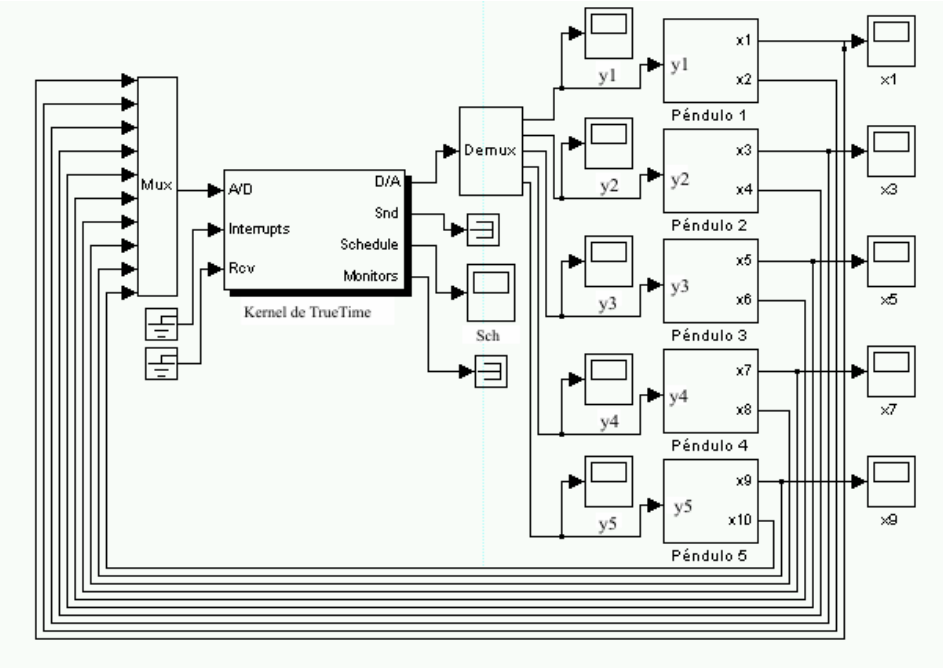

Figura 15. Esquema de la Aplicación de control implementada con TrueTime.

El kernel de TrueTime envía las cinco acciones de control a través del conversor digital-analógico (D/A) a cada uno de los péndulos. La salida de cada

péndulo es realimentada al kernel de TrueTime a través del conversor analógico-digital (A/D) para calcular la siguiente acción de control.

El periodo de muestreo menor es el del péndulo 1 y consecutivamente el periodo aumenta hasta el péndulo 5 que es el que posee el mayor periodo de muestreo. Por consiguiente, a la tarea de control del péndulo 1 se le asignó la mayor prioridad y en forma consecutiva hasta la tarea de control del péndulo 5 a la que se le asignó la menor prioridad en la banda superior. Esta asignación se realizó pues es óptima en una diagramación de prioridades fijas ([30]).

Cada uno de los sistemas de tiempo real fue simulado considerando planificadores basados en Prioridades Fijas (FP) y Menor Tiempo al Vencimiento (EDF) para comparar con la eficiencia del planificador PBJ.

Las Figuras 16, 17, 18, 19 y 20 muestran el promedio de los tiempos de establecimiento de cada péndulo obtenidos con cada disciplina. El tiempo de establecimiento fue elegido como figura de comparación debido a la estrategia de control vertical, como se menciono en el capítulo 6, es muy sensible al jitter.

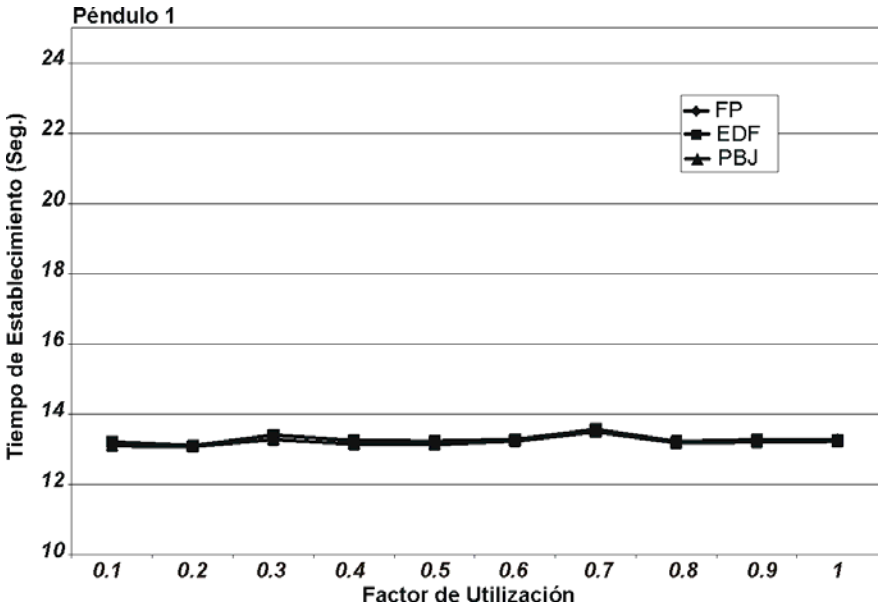

Figura 16. Tiempo de establecimiento del péndulo 1 usando cada disciplina de prioridad

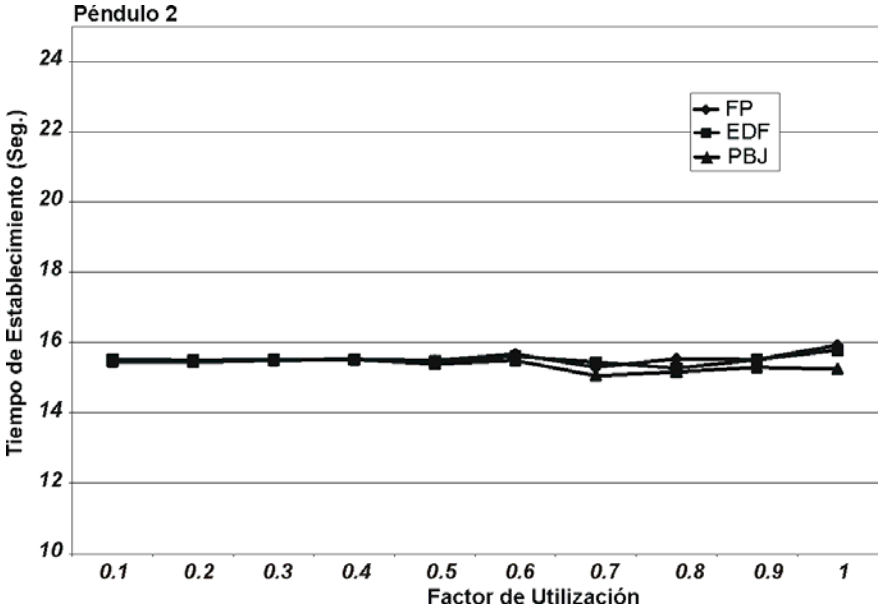

Figura 17. Tiempo de establecimiento del péndulo 2 usando cada disciplina de prioridad

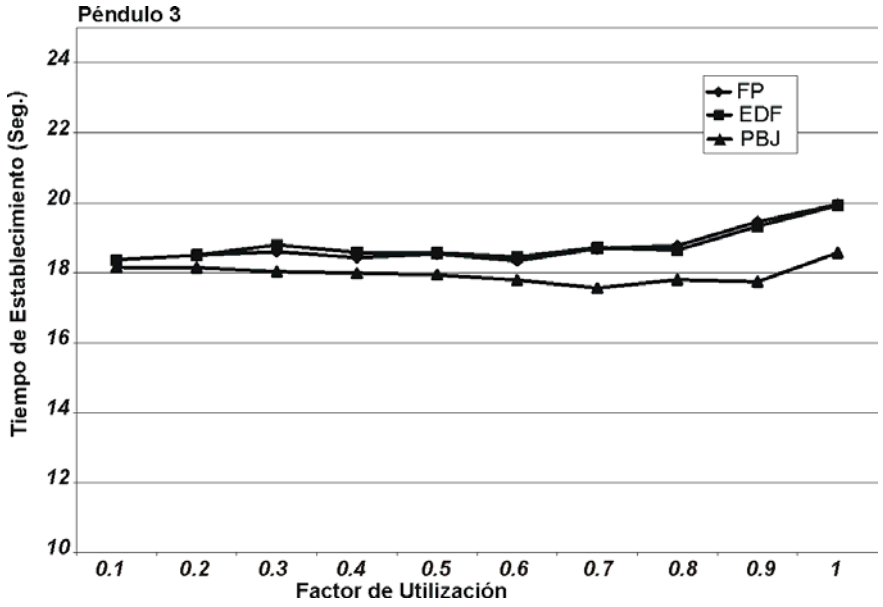

Figura 18. Tiempo de establecimiento del péndulo 3 usando cada disciplina de prioridad

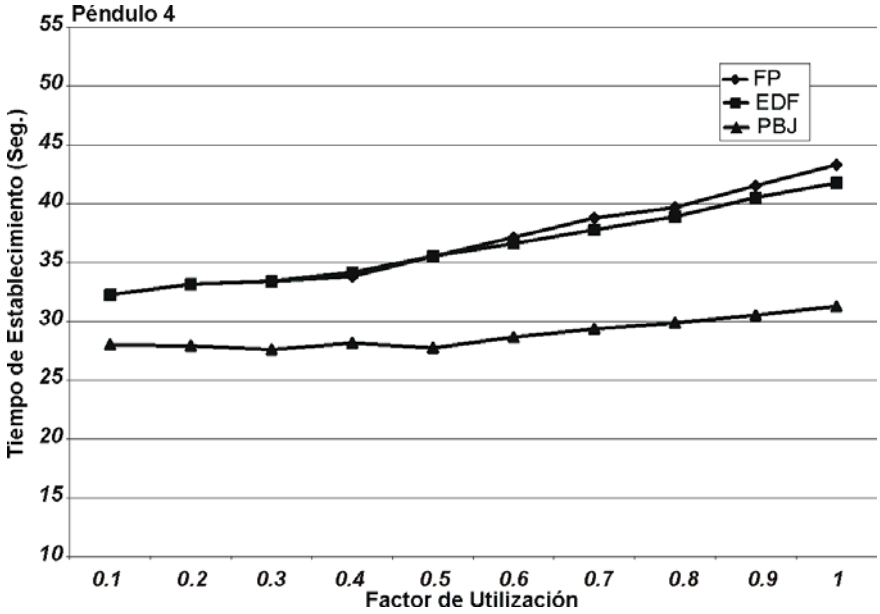

Figura 19. Tiempo de establecimiento del péndulo 3 usando cada disciplina de prioridad

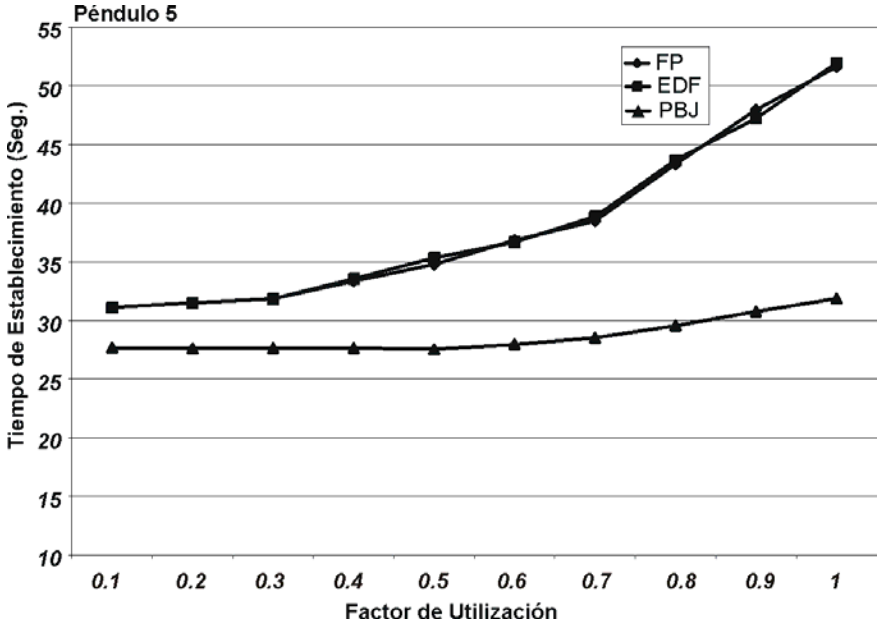

Figura 20. Tiempo de establecimiento del péndulo 3 usando cada disciplina de prioridad

7.5 Análisis y resultados

Notamos y confirmamos, a través de las simulaciones, que un jitter mayor al 20% no fue tolerado por las estrategias verticales, dejando a los péndulos en un estado de oscilación de manera que nunca alcanzan la posición vertical. Notamos

también que el desempeño de PBJ es mucho mejor que el de Prioridades Fijas y EDF para cada uno de los péndulos de la aplicación.

El desempeño de las estrategias de control para los péndulos uno y dos es similar para los tres mecanismos. Esto se debe a que estas tareas de control son asignadas con las mayores prioridades a lo largo de un tiempo prolongado de la vida del sistema. Tanto en EDF como en FP, estas tareas de control son ejecutadas con bajo jitter cuando el factor de utilización es bajo.

Observamos también que las tareas de control necesitaron ser promovidas a la banda superior apenas en el 15% de las invocaciones. De esta manera, el desempeño de tiempo real de las tareas que no son de control es mejorado. La Figura 21 muestra el tiempo de respuesta medio de acuerdo al período de muestreo de las tareas que no son de control para cada factor de utilización obtenido con cada disciplina.

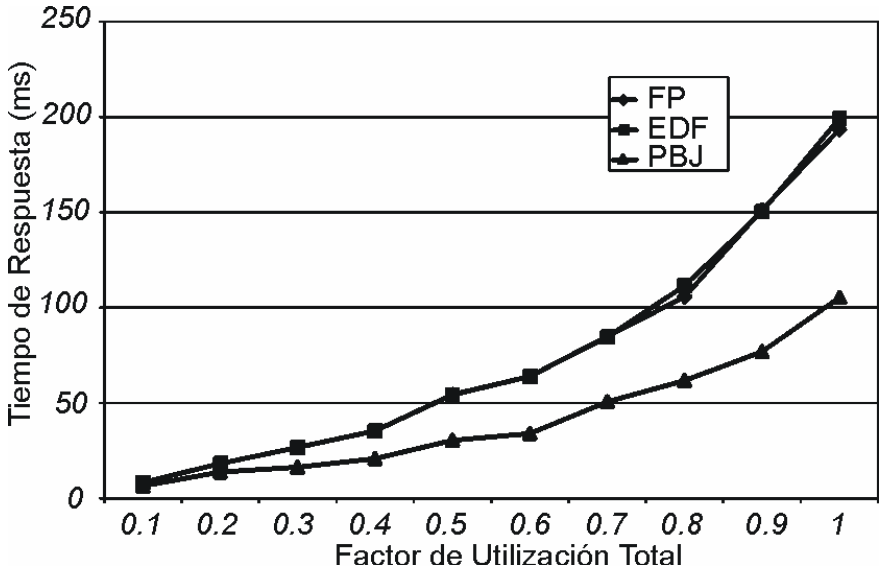

Figura 21. Tiempo de respuesta medio de tareas que no son de control obtenido con cada disciplina.

Observamos que con el planificador PBJ las tareas que no son de control mejoran su tiempo de respuesta considerablemente en relación a PF y EDF a medida que el factor de utilización aumenta.

Por otro lado, se obtuvo el tiempo de establecimiento de cada péndulo cuando su estrategia de control es la única tarea corriendo en el sistema de tiempo real. Los resultados para cada péndulo son mostrados en la Tabla 1. El tiempo de

establecimiento de cada péndulo, depende de las características de control de cada uno de ellos relacionadas sobre todo con sus características constructivas.

Tabla 1. Tiempo de establecimiento de las estrategias de control de cada péndulo corriendo solas en un sistema de tiempo real

Péndulo	Tiempo de Establecimiento (ms)
1	13
2	15
3	16.91
4	25.21
5	25.3

Como podemos observar, el valor obtenido con el planificador PBJ es muy cercano al valor obtenido en la Tabla 1.

7.6 Conclusiones

En este Capítulo se propuso un mecanismo de prioridades que reduce los efectos adversos que la planificación de tiempo real produce sobre aplicaciones de control. El planificador PBJ está basado en la promoción de tareas de control a la banda de prioridad superior de acuerdo a la sensibilidad que tales tareas presentan al jitter en tiempo de ejecución.

Evaluamos su desempeño usando TrueTime en un ambiente Matlab/Simulink y lo comparamos con los obtenidos con Prioridades Fijas y EDF. Mostramos que el planificador PBJ mejora el desempeño de control de la aplicación y, al mismo tiempo, el tiempo de respuesta de las tareas que no son de control.

Por otra parte, pudimos comprobar en muchos casos, que EDF y PF no son capaces de estabilizar algunos péndulos aún cuando los sistemas de tiempo real eran planificables mientras que PBJ fue capaz de estabilizarlos.

Capítulo 8

Utilización del uRT51 para Planificación de Tareas de Control

En esta sección se describen y evalúan las principales características del procesador uRT51. Éste fue diseñado para aplicaciones embebidas de control de tiempo real. La arquitectura del procesador incorpora las funciones específicas de un sistema de tiempo real en hardware. El procesador está descrito en VHDL y es apto para ser implementado en diferentes FPGAs. En esta sección también se describirá cómo el procesador uRT51 soporta tiempo, eventos, tareas y prioridades. El desempeño del mismo es evaluado usando una aplicación como caso de estudio. Los experimentos demuestran que las características de planificación del procesador uRT51 superan las obtenidas por sistema tradicional de tiempo real, RTOS.

8.1 Introducción

La diversidad de los sistemas de tiempo real incluye telefonía celular, dispositivos de entretenimiento, industria automotriz, juguetes, tarjetas

electrónicas, dispositivos para medicina, equipamiento para redes, sensores y robots industriales. Estos son modelados como un conjunto de tareas que tienen que ser ejecutadas por un procesador antes de un determinado vencimiento.

Los sistemas de tiempo real embebidos son usualmente construidos utilizando Sistemas Operativos de Tiempo Real (RTOS) o diseñando todo el sistema como un programa monolítico basado en recursos de hardware específicos. Los sistemas operativos de tiempo real permiten que el diseñador alcance un alto nivel de abstracción y portabilidad, además de mejorar la verificación y las características de mantenimiento del sistema. En la literatura podemos encontrar varios RTOSs que han sido diseñados para dar soporte a aplicaciones de tiempo real (ej.: estándar POSIX ([28]), spring kernel ([41]), QNX ([27])). Cada RTOS incluye una tarea especial del sistema, denominado *planificador,* el cual comparte el procesador con tareas que requieren ejecución. El planificador implementa una disciplina de prioridades para determinar la siguiente tarea a ser ejecutada. Además debe ser ejecutado periódicamente, produciendo sobrecarga en el sistema.

Algunas aplicaciones pueden requerir características temporales estrictas o pueden no tolerar la sobrecarga en tiempo de ejecución producido por el sistema operativo. En estos casos, es obligatorio un uso directo de los recursos del hardware. Los timers, contadores e interrupciones tienen que ser directamente programados dificultando el mantenimiento del sistema.

El uRT51 es un procesador para aplicaciones embebidas de tiempo real. Su arquitectura fue diseñada desde el punto de vista de tiempo real y puede soportar diferentes disciplinas de prioridad sobre un conjunto de más de 65000 tareas. El uRT51 es escalable y consecuentemente el número soportado de tareas y eventos depende de la memoria física del sistema. Está descripto en VHDL y puede ser usado para construir arquitecturas en chips (SoC, systems-on-chip) sobre dispositivos lógicos programables FPGA. En el uRT51, los parámetros temporales de cada tarea, como así también la disciplina de prioridades pueden ser definidos independientemente de la funcionalidad de cada una de ellas. Por lo tanto, el código de las tareas no incluye la configuración de sus características de tiempo real, lo cual permite un alto nivel de abstracción en la implementación y una mayor flexibilidad en el diseño ([40]).

En las siguientes secciones de este capítulo, describimos las características principales del procesador uRT51 y evaluamos la eficiencia de éste en una aplicación de control que consiste en el control de velocidad de un motor de DC. Al final del capítulo concluimos, por medio de los experimentos, que el uRT51 es mucho más eficiente que un sistema embebido de tiempo real basado en RTOS.

8.2 La arquitectura del uRT51

En esta sección se describen los componentes principales de la arquitectura del procesador uRT51. En [40], se puede obtener una descripción más completa de las características del uRT51.

La Figura 22 muestra un esquema de un sistema basado en un procesador uRT51 que consiste de:

- El núcleo 8051: Éste implementa un subconjunto de las instrucciones del procesador 8051 y puede reconocer algunas instrucciones especiales de tiempo real para configurar el Administrador de Tiempo Real. Los dispositivos de Entrada/Salida pueden ser directamente manejados por el núcleo 8051.
- El Administrador de Tiempo Real: Éste controla el comportamiento en tiempo real del procesador uRT51. Está continuamente verificando si existe alguna acción de tiempo real para ejecutar. Cuando no hay tareas requiriendo ser ejecutadas, el administrador de tiempo real detiene todas las actividades del uRT51 y por consiguiente se reduce el consumo de energía del sistema.
- La Unidad de Depuración: Ésta permite una fácil integración con la plataforma de desarrollo del uRT51 (uRT51 Programming Suite). Todas las actividades del uRT51 pueden ser controladas y supervisadas por plataforma de desarrollo del uRT51 a través de la Unidad de Depuración.
- El Controlador de Memoria: Éste implementa toda la funcionalidad para conectar el procesador uRT51 con memorias externas. Soporta diferentes anchos de canal.

- El Administrador de Interrupciones: Éste da soporte a eventos de tiempo real asincrónicos que pueden generar requerimientos de ejecución de tareas de tiempo real.

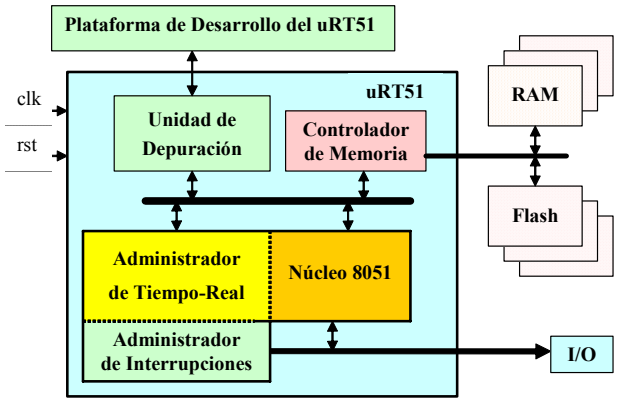

Figura 22. Layout de la arquitectura del uRT51.

La memoria RAM es compartida entre el núcleo 8051 y el Administrador de Tiempo Real. El Administrador de Tiempo Real almacena toda la información de tiempo real que éste necesita para ejecutar las tareas de tiempo real en la RAM del sistema. Esta información incluye tiempo de generación, vencimiento, prioridad y estado de cada tarea de tiempo real. El Administrador de Tiempo Real comprueba si existe un evento que fuerza el cambio de estado del núcleo 8051.

El Administrador de Tiempo Real no es una unidad de procesamiento, éste es un circuito digital optimizado para realizar las funciones de tiempo real del procesador uRT51. La arquitectura fue diseñada para trabajar con un reloj externo de 10 MHz. Con esta frecuencia, tanto el período como el vencimiento de la tarea pueden ser expresados con una resolución de tiempo de 100 nseg. El núcleo 8051 no ejecuta ninguna subrutina timer y por lo tanto el sistema de tiempo real no es sobrecargado. De este modo, tanto la planificabilidad del sistema como el tiempo de respuesta de las tareas de tiempo real se mejoran.

8.2.1 Tiempo y Eventos en el uRT51.

Un procesador de tiempo real debe soportar tiempo adecuadamente para poder atender los eventos que van ocurriendo. Definimos un *evento* como una ocurrencia o suceso, generalmente significativo a la ejecución de una función, operación, o tarea. Por consiguiente, la evolución de un sistema de tiempo real

puede ser definida adecuadamente expresando el tiempo en el que ocurren los eventos. Generación, vencimiento, fin de ejecución, promoción de prioridades, aborto de ejecución son algunos ejemplos de eventos en un sistema de tiempo real.

El procesador uRT51 soporta tiempo y eventos adecuadamente. Éste mantiene el tiempo del sistema en registros internos del Administrador de Tiempo Real. Éste verifica continuamente si existe un evento que requiera ser activado o una tarea cuyo estado deba ser modificado. La eficiencia del manejo de información en tiempo real es alta y confiable debido a que los eventos y las tareas son manejados independientemente de la ejecución de las tareas de tiempo real. Además, el jitter y la eficiencia en tiempo real del uRT51 pueden ser medidos en períodos del reloj del sistema en vez de ser medidos usando el intervalo de tiempo definido por el período del planificador (T_{timer}), como es el caso en sistemas basados en RTOSs. Por lo tanto, la eficiencia es mejorada en varios órdenes de magnitud. El máximo número de eventos soportados, depende la capacidad física de memoria de la Memoria de Tiempo Real asignada a las estructuras de eventos.

8.2.2 Tareas y Prioridades.

Debido a que muchos modelos de procesos de tiempo real consideran la ejecución en un procesador de un conjunto de tareas, una arquitectura de tiempo real debe soportar múltiples tareas. Por consiguiente, una arquitectura de tiempo real tiene que manejar todos los parámetros de la tarea necesarios para proporcionar un ambiente de múltiples tareas. El procesador uRT51 utiliza *estructuras de tareas* para almacenar los parámetros de las tareas de tiempo real. Las estructuras de tareas son almacenadas en la memoria RAM del sistema. El número máximo de tareas soportadas, depende, nuevamente, de la cantidad física de memoria asignada a estructuras de tareas en la Memoria de Tiempo Real.

En un sistema multitarea se asigna una prioridad a cada tarea para planificar el conjunto de tareas que están listas para ser ejecutadas. La prioridad de cada tarea puede ser modificada así como también mantenida fija en tiempo de ejecución de acuerdo a la disciplina de planificación implementada.

Aproximaciones previas de procesadores de tiempo real implementan una disciplina de planificación predefinida sobre un conjunto reducido de prioridades ([19], [20], [18], [1], [34]). El procesador uRT51 no ejecuta una disciplina de prioridad predeterminada sino que elige para la ejecución la tarea de mayor prioridad. La prioridad de cada tarea es mantenida en la estructura de tarea respectiva y ésta puede se modificada en tiempo de ejecución de acuerdo a la configuración de tiempo real realizada.

La disciplina de planificación depende de cómo es asignada la prioridad a cada tarea. Esta prioridad se puede modificar configurando la acción requerida en eventos o interrupciones temporizadas. Por el contrario, si los eventos del sistema no modifican la prioridad de la tarea, entonces es implementada una disciplina de prioridades fijas. Por lo tanto, pueden ser implementadas diferentes disciplinas de prioridades.

8.2.3 Configuración de Tiempo Real.

El Administrador de Tiempo Real debe ser configurado para activar los parámetros de tiempo real de la aplicación de tiempo real. Dicha configuración debe ser realizada por la ejecución de instrucciones de tiempo real especiales. El conjunto de instrucciones de tiempo real amplía el conjunto original de instrucciones del núcleo del 8051. Cuando es configurado el Administrador de Tiempo Real, rara vez es necesario ejecutar instrucciones de tiempo real adicionales. No obstante, las tareas de tiempo real pueden ejecutar instrucciones de tiempo real en tiempo de ejecución.

Una vez que el Administrador de Tiempo Real es configurado, éste realiza todas las funciones de tiempo real al mismo tiempo que el núcleo del 8051 ejecuta el código de las tareas de tiempo real.

El conjunto completo de instrucciones de tiempo real puede ser encontrado en [40]. Sin embargo, la plataforma de desarrollo del uRT51 le muestra al usuario una interfaz amigable para configurar el procesador uRT51 sin ningún conocimiento de las instrucciones de tiempo real.

8.3 Plataforma de desarrollo del uRT51

La plataforma de desarrollo del uRT51 es una colección de herramientas completamente equipada, de alto rendimiento, interactiva y fácil de usar que soporta programación, depuración y análisis de sistemas de tiempo real implementados sobre el procesador uRT51. Esta plataforma incluye todas las facilidades para el análisis en tiempo de ejecución y soporta registro de datos. En la siguiente sección se usa dicha plataforma para la implementación del caso de estudio de este capítulo.

La plataforma de desarrollo del uRT51 describe un sistema de tiempo real en términos de tareas, propiedades de tiempo real y disciplinas de prioridades. El usuario puede agregar tantas tareas como desee dentro de las limitaciones de memoria ([40]).

El principal objeto de un sistema de tiempo real en la plataforma de desarrollo del uRT51 es una tarea de tiempo real. Una tarea realiza una cierta función de la aplicación. Está es definida por:

- Su código: éste es programado de acuerdo a la función que la tarea debe realizar.
- Sus propiedades de tiempo real: son las características temporales tales como invocación periódica, prioridad, vencimiento, etc. Alguna de las propiedades de tiempo real depende de la disciplina de prioridad seleccionada.

El código define la manera en que la tarea realiza su función y las propiedades de tiempo real establecen el comportamiento de ésta en tiempo de ejecución. La plataforma de desarrollo del uRT51 incluye un Editor de Tarea para programación concurrente tanto de la funcionalidad como de las características de tiempo real de la aplicación.

8.4 Experiencias Realizadas

Se implementó el sistema procesador uRT51 sobre una plataforma de desarrollo FPGA Excalibur basado en un dispositivo APEX EP20FE200 de Altera. La memoria requerida para el procesador uRT51 fue implementada usando

el EAB incluido en la arquitectura APEX. La arquitectura entera requiere 4500 LEs aproximadamente.

Se implemento un planificador típico de RTOS para ser ejecutado sobre el núcleo 8051 del uRT51 con el objetivo de compara eficiencias entre ambas implementaciones. Se lo implementó como una rutina de interrupción temporizada (timer) que no ejecuta ninguna instrucción de tiempo real de la unidad Administrador de Tiempo Real del uRT51. Cuando se considera un sistema de tiempo real basado en un RTOS, el intervalo del timer debe ser definido y el período de cada tarea expresado en unidades de tiempo de dicho intervalo denominado *slot*. Los intervalos del timer fueron seleccionados para obtener períodos de las tareas entre 2 slots y 255 slots (de lo contrario se debe implementar una aritmética de 16-bit sobre un procesador de 8-bit).

Las experiencias fueron realizadas considerando frecuencias de reloj del sistema iguales a 10, 20, 40, 50, 80 y 100MHz. La plataforma de desarrollo del uRT51 fue utilizada para programar el sistema y para analizar su comportamiento en tiempo de ejecución.

8.4.1 Caso de Estudio: Control de velocidad de un motor de DC

En esta sección se describe el control de velocidad de un motor de DC el cual es usado para evaluar la eficiencia del procesador uRT51.

La velocidad del motor de DC es controlada variando la energía transferida al motor usando la técnica de modulación del ancho de pulso (PWM). La medición de la misma se realiza indirectamente usando un sensor óptico y un disco ranurado unido al eje del motor (Figura 23). Por lo tanto, contando el número de ranuras ocurridas en un cierto intervalo I_s, se obtiene la medición requerida.

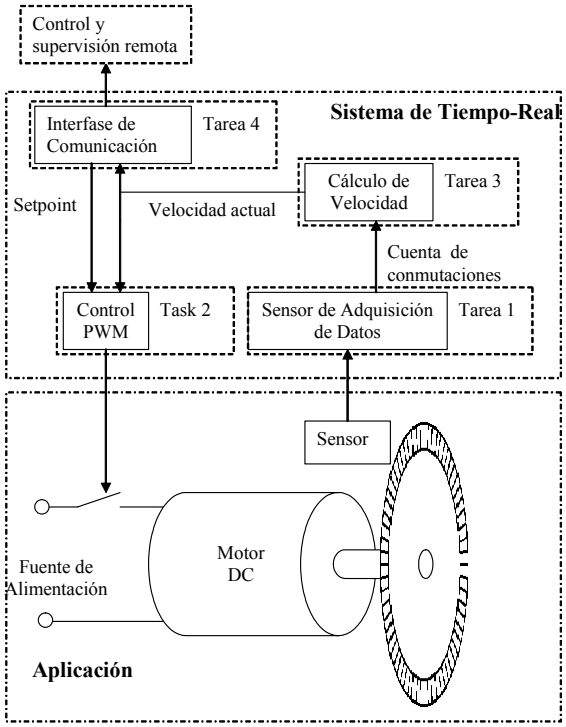

Figura 23. Esquema del control de velocidad del motor de DC

Podemos hacer el sistema más predecible implementando la tarea sensor como una tarea periódica de tiempo real en lugar de usar contadores especiales en rutinas de interrupción ([31]). De esta manera, nos aseguramos que podemos incluir este sistema como parte de una aplicación más grande y, si el sistema es planificable, entonces la aplicación de control trabajará apropiadamente. De otra manera, debemos asegurarnos que las rutinas de interrupción no interfieran con las otras tareas de tiempo real del sistema.

Pueden definirse cuatro tareas de tiempo real desde las funciones de control que el sistema debe realizar:

- **Tarea 1**: Adquisición de Datos. Esta tarea se encarga de monitorear el sensor y de contar el número de conmutaciones que éste produce sobre su entrada.
- **Tarea 2**: Control PWM. Esta modifica el ciclo de servicio de la onda PWM de acuerdo a la diferencia entre el setpoint y la velocidad del motor.
- **Tarea 3**: Cálculo de Velocidad. Esta tarea calcula la velocidad del motor por medio del ángulo transcurrido en el intervalo I_s.

- **Tarea 4**: Interfaz de Comunicación. Esta tarea envía y recibe información de control y monitoreo de un supervisor de datos por medio de un enlace serie asincrónico trabajando a 9600 bps.

8.4.2 Implementación y Evaluación.

Mientras los períodos y vencimientos dependen de las funciones que las tareas de control deben realizar, el tiempo de ejecución depende de las características computacionales del procesador. El peor tiempo de ejecución de cada tarea es dado en la Tabla 2 en unidades del período de reloj del sistema.

Tabla 2. Tiempo de ejecución del peor caso de las tareas del sistema expresados en períodos del reloj

Tarea	*C* [períodos de reloj]
1	401
2	436
3	506
4	366

La asignación de prioridades es realizada de acuerdo a la política de Prioridades Fijas. Tareas con menor periodo son asignadas con mayor prioridad. La Tabla 3 muestra el período, el vencimiento y la prioridad de cada una de las tareas de tiempo real. Los periodos de las tareas fueron elegidos de acuerdo a las propiedades de control de la aplicación. Las prioridades mayores son representadas por los índices de prioridad menores.

Tabla 3. Período, vencimiento y prioridad de las tareas de tiempo real.

Tarea	T	D	Prioridad
1	347.22μs	347.22μs	0 (mayor)
2	1*m*s	1*m*s	1
3	23*m*s	23*m*s	3
4	1*m*s	1*m*s	2

Comparamos ambas implementaciones (RTOS vs. uRT51) midiendo el jitter de cada uno de ellos. Como se ha mencionado antes, cuando las tareas de control son afectadas por jitter, frecuencias de salida indeseables son aplicadas a las aplicaciones de control. Estas frecuencias indeseables son transmitidas a la aplicación a través de los actuadores y pueden producir efectos no buscados tales

como: vibraciones, calentamiento, inestabilidad, ruido entre otros. Las Figuras 24, 25, 26 y 27 muestran el jitter promedio de la tarea 1, 2, 3 y 4 respectivamente.

Desde las experiencias, obtenemos que el procesador uRT51 puede planificar el sistema con una frecuencia de reloj de 10MHz, mientras que el sistema basado en RTOS necesita al menos una frecuencia de reloj de 50MHz. Además, la eficiencia de control del procesador uRT51 con un reloj de 10MHz no puede ser mejorada ni siquiera usando un RTOS con un reloj de 100MHz.

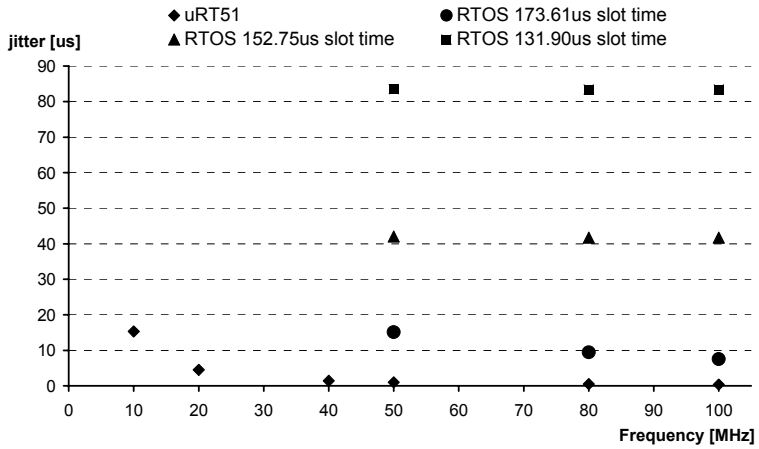

Figura 24. Jitter promedio de la tarea 1

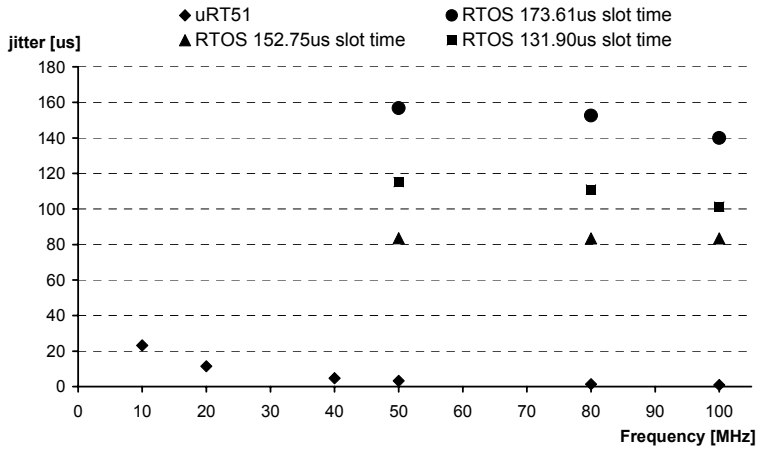

Figura 25. Jitter promedio de la tarea 2

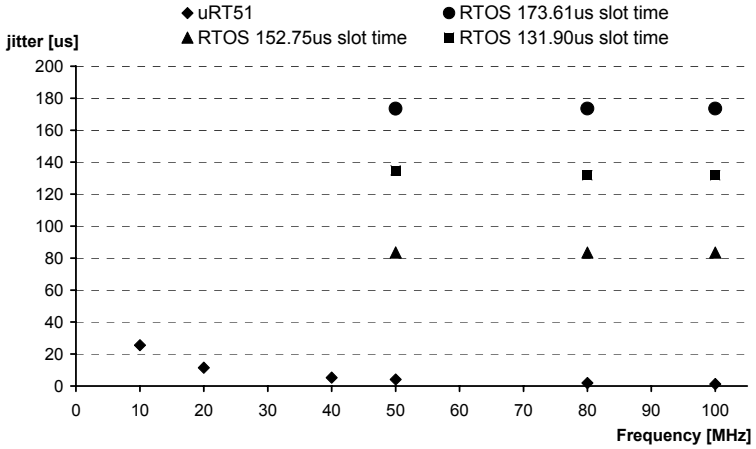

Figura 26. Jitter promedio de la tarea 3

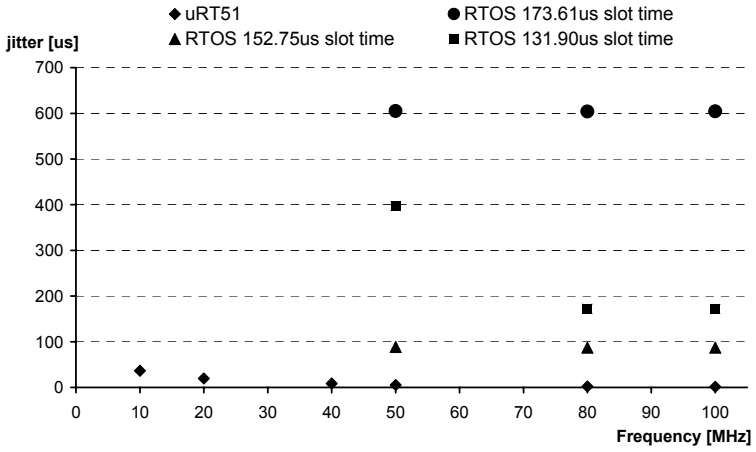

Figura 27. Jitter promedio de la tarea 4

Las figuras muestran que la eficiencia de un sistema basado en RTOS es muy sensible al tiempo de slot elegido para ejecutar el planificador. Esto hace el diseño complicado cuando el sistema es construido desde un conjunto de subsistemas de tiempo real.

Por otra parte, el procesador uRT51 no introduce ningún tiempo de slot y, en consecuencia, su eficiencia sólo depende de la frecuencia de reloj del sistema.

8.5 Análisis y resultados

Desde el punto teórico de planificación de tiempo real, la sobrecarga del sistema es reducida en una implementación basada en uRT51 y el sistema es planificable a una frecuencia de reloj baja con las siguientes ventajas:

- El consumo de energía del sistema es reducido. Debido a que el sistema puede ser planificado a baja frecuencia de reloj, el voltaje y consecuentemente el consumo de energía del sistema del sistema puede ser dramáticamente reducido. Por otra parte, el uRT51 elude la sobrecarga introducida por el RTOS y como resultado de ello el consumo de energía es también reducido.
- Las tecnologías de bajo costo pueden ser utilizadas. Cuando son aplicadas bajas frecuencias de reloj, pueden utilizarse tecnologías de menor costo para la implementación del sistema. Por lo tanto, debería necesitarse un dispositivo más sofisticado si es implementado un sistema basado en RTOS.

Por otra parte, desde el punto de vista de la aplicación, la eficiencia es mejorada cuando es utilizado un procesador uRT51 debido a que el jitter promedio de las tareas es mucho menor. Esta característica puede ser muy importante cuando el sistema de tiempo real es aplicado a aplicaciones de control.

En un sistema de tiempo real basado en RTOS, el tiempo de slot puede tener una mayor influencia sobre la eficiencia de control que la frecuencia del reloj del sistema. Incrementando la frecuencia del reloj en un sistema basado en RTOS, puede no disminuir en la misma proporción el jitter introducido. Puede no haber un tiempo óptimo de slot para todas las tareas: en nuestro caso de estudio, 173,61 μs es óptimo para la tarea 1, mientras que un tiempo de slot de 152,75 μs es óptimo para la tarea 2, 3 y 4. Por otra parte, podemos observar que debido a que el Administrador de Tiempo Real es un circuito digital (no una unidad procesadora), el jitter en un procesador uRT51 es simplemente inversamente proporcional a la frecuencia del reloj.

8.6 Conclusiones.

En éste capítulo se describieron las principales características del procesador uRT51. Se implementó el control de velocidad de un motor de DC para evaluar experimentalmente la eficiencia de procesador en una aplicación de control y se la comparó con la eficiencia de un sistema operativo de tiempo real.

La eficiencia del procesador uRT51 permite planificar el sistema de tiempo real a muy baja frecuencia. Mientras el procesador uRT51 planifica la aplicación del control de velocidad del motor de DC a 10MHz de reloj, el sistema basado en RTOS necesita al menos 50MHz de reloj.

Debido a que el jitter de las tareas de control es reducido, las propiedades de control del sistema de tiempo real son mejoradas. Por lo tanto, las perturbaciones que el sistema de tiempo real produce sobre la aplicación de control son reducidas también. Puede alcanzarse una mayor precisión en las características temporales del sistema.

El diseño de una aplicación de tiempo real con el procesador uRT51 se puede realizar en un alto nivel de abstracción debido a que la funcionalidad de cada tarea puede ser definida independientemente de sus características de tiempo real. Por lo tanto, las características temporales de cada tarea pueden ser modificadas sin reprogramar su código, a diferencia del caso en el que se utilizara un RTOS.

Podemos concluir en que la arquitectura del procesador uRT51 es adecuada para sistemas de propósito dedicado de tiempo real en aplicaciones de control.

La utilización del procesador uRT51 permite reducir las perturbaciones que el sistema de tiempo real introduce en la aplicación de control a las que las tareas producen entre sí, evitando cualquier otra perturbación proveniente de la ejecución del sistema operativo. Como se mostró mediante la evaluación realizada en este capítulo, las perturbaciones que introduce un sistema operativo de tiempo real no son insignificantes, aunque generalmente se las desprecia. De esta manera, todos los resultados obtenidos mediante los mecanismos propuestos en esta tesis podrán ser realmente logrados mediante una implementación basada en una arquitectura uRT51.

Capítulo 9

Control de Sistemas Complejos con Mecanismos Flexibles

En esta sección se desarrolla un mecanismo que adapta la carga que las tareas de control producen sobre un sistema de tiempo real sin degradar su eficiencia de control. Este mecanismo es basado en el desarrollado por Martí en [33], con la principal diferencia de que la variación del período de muestreo se la realiza en forma progresiva para controlar las perturbaciones en aplicaciones de control. Sobre el final de esta sección se compara el mecanismo desarrollado con los desarrollados en [14] y [33].

9.1 Introducción.

Como se mencionó anteriormente, un sistema de control de tiempo real típico está constituido de varias actividades de control en conjunto con otras operaciones que no lo son. Estas operaciones necesitan ser ejecutadas periódicamente y dentro de un estricto límite de tiempo para garantizar que el sistema cumpla con los requerimientos de desempeño requeridos.

Uno de los problemas más importantes de los sistemas de control de tiempo real es la carga que le introducen cada una de las tareas que corren en él. En esta sección proponemos un mecanismo que adapta la carga que una tarea de control produce sobre un sistema de tiempo real sin degradar su eficiencia de control.

Todos los trabajos ya mencionados ([16], [4], entre otros) han analizado los efectos que las políticas de planificación producen sobre las aplicaciones de control. Éstos proponen diferentes modelos de tareas y algoritmos para reducir el jitter, pero ninguno de ellos evalúa la eficiencia de control cuando se implementan estrategias de control complejas.

En [23], [14] y [33] se proponen algoritmos adaptivos on-line para cambiar el período de la tarea de acuerdo a la carga del sistema. Sin embargo, no se considera la degradación que producen sobre el desempeño de control.

El mecanismo propuesto en este capítulo es basado en el mecanismo propuesto por Martí en [33]. Introducimos una variación progresiva del período de la tarea para controlar la perturbación producida en la aplicación de control.

La idea general del mecanismo es detectar cuándo el factor de utilización del procesador aumenta por encima de un cierto umbral. Cuando esto ocurre, el período de la tarea de control es incrementado para reducir el factor de utilización total con la precaución de mantener la planificabilidad del sistema de tiempo real. La eficiencia de control puede ser degradada cuando el período de la tarea de control es aumentado. Por este motivo, en el mecanismo propuesto se tiene en cuenta una solución de compromiso entre las eficiencias de tiempo real y de control.

9.2 Variación del período de muestreo.

La selección del período de muestreo es muy importante para el desarrollo de este mecanismo. Como se mencionó en la Sección 2.2 de Capítulo 2, el período de muestreo puede ser elegido dentro de un rango determinado. Por otro lado, tendiendo en cuenta el análisis introducido en [37], podemos obtener un período de muestreo óptimo, dentro de este rango, el cual lo llamaremos T^{opt}

Cambiando el período de una tarea de control en tiempo de ejecución pueden producirse efectos indeseados sobre la aplicación de control. Las estrategias de

control necesitan de las entradas previas para calcular la salida de control, las cuales deben ser periódicamente muestreadas. Cuando el período de la tarea de control es modificado, las muestras deben ser tomadas con el nuevo periodo. Sin embargo, las muestras almacenadas en memoria fueron tomadas con el período previo y por consiguiente pueden introducir un error sobre la estrategia de control. Este error aumenta proporcionalmente con las diferencias entre los períodos, el rango dinámico de la entrada y el número de muestras previas usadas por la estrategia de control.

El mecanismo propuesto en esta sección, cambia gradualmente el período de la tarea de control paso a paso desde el período inicial hasta alcanzar el período final. Después de cada modificación del período, el mecanismo permite que la estrategia de control actualice las muestras almacenadas en la memoria.

Por ejemplo, si la aplicación de control comienza a ser muestreada con $T^0 = T^{opt}$ y el factor de utilización del sistema de tiempo real es mayor que un determinado umbral, entonces el mecanismo aumenta el período de la tarea de control desde T^0 a T^i con un paso n, como sigue:

$$T^0 = T^{opt} \rightarrow T^1 = T^0 + n \rightarrow T^2 = T^1 + n \rightarrow \ldots \rightarrow T^i = T^{i-1} + n \quad (\mathbf{22})$$

donde n es la máxima cantidad que el período de muestreo puede ser cambiado en cada paso. El incremento n puede ser determinado por la sensibilidad de la estrategia de control al error en la entrada expresado como:

$$n = \max(\Delta T) \text{ tal que } \gamma \cdot \Delta T \cdot \max \frac{de}{dt} < \varepsilon_{act} \quad (\mathbf{23})$$

donde ε_{act} es la precisión del actuador entregado por el fabricante, γ es una constante de proporcionalidad y de/dt es la velocidad de modificación de la entrada del controlador.

9.3 Implementación del mecanismo propuesto

Los resultados presentados en [33] del mecanismo que modifica el período de muestreo de la tarea de control en tiempo de ejecución, provienen de simulaciones

realizadas en Matlab/Simulink las cuales no han modelado adecuadamente los efectos de la implementación real. En [33], el efecto de muestreo no fue tenido en cuenta y por consiguiente Matlab produce el correcto muestreo de la entrada debido a que ésta es considerada una señal continua. Éste no es el caso real, ya que cuando el sistema es implementado, la entrada ya no es continua y la modificación del período de muestreo introduce un error en el muestreo.

El mecanismo propuesto acota el error introducido al modificar el período de muestreo en tiempo de ejecución. El efecto real del muestreo es adecuadamente modelado en Matlab.

La Figura 28 muestra los pasos a seguir para calcular la estrategia de control en un sistema de tiempo real.

```
read(y);
u = calcuate(x_k, -L(T));
write(u_k);
x_{k+1} = update(x_k, Φ_cL(T));
```

Figura 28. Cálculo de la estrategia de control sin adaptación de carga

El mecanismo propuesto es introducido en la Figura **29**.

```
read(y);
obtain(T_k);
calculate(Φ(T_k),Γ(T_k),L(T_k), Φ_cL(T_k));
u = calcuate(x_k, -L(T));
write(u_k);
x_{k+1} = update(x_k, Φ_cL(T));
```

Figura 29. Cálculo de la estrategia de control con adaptación de carga

La instrucción **obtain(T_k)** retorna el período actual de la tarea de control y éste es usado para determinar los parámetros de control actuales con la instrucción **calculate($\Phi(T_k), \Gamma(T_k), L(T_k), \Phi_{cL}(T_k)$).** El período de la tarea de control es modificado por el planificador de acuerdo a la carga actual del sistema de tiempo real.

9.4 Caso de estudio. Sistema dos masas y resorte.

Para mostrar las ventajas de este mecanismo, como caso de estudio, presentamos un sistema típico en la literatura de control constituido por dos masas unidas a través de un resorte ([46]). El mismo se presenta en la Figura 30

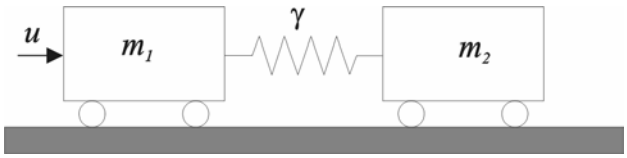

Figura 30**.** Sistema de dos masas unidas por un resorte.

La acción de control u actúa sobre la primer masa así la segunda masa alcanza su posición final tan rápido como sea posible. Si el peso de las masas (m_1 y m_2) y la constante del resorte (γ) son normalizadas a uno, la función de transferencia a lazo abierto del sistema está dada por:

$$G(s)=\frac{1}{s^4+2\cdot s^2} \qquad (24)$$

Para estabilizar este sistema obtenemos el siguiente controlador de tercer-orden (para más detalle ver [26]).

$$C(s)=\frac{1+4{,}578\cdot s+0{,}1094\cdot s^2-1{,}9687\cdot s^3}{4{,}9688+8{,}0469\cdot s+4{,}5781\cdot s^2+s^3} \qquad (25)$$

Aplicando la transformación bilineal,

$$s=\frac{2}{T}\left(\frac{z-1}{z+1}\right) \qquad (26)$$

obtenemos la siguiente función de transferencia discreta,

$$C(z)=\frac{b_0\cdot z^3+b_1\cdot z^2+b_2\cdot z+b_3}{a_0\cdot z^3+a_1\cdot z^2+a_2\cdot z+a_3} \qquad (27)$$

desde la cual obtenemos la siguiente acción de control.

$$u(k)=\frac{1}{a_0}\begin{bmatrix} b_0e(k)+b_1e(k-1)+b_2e(k-2)+b_3e(k-3)- \\ -a_1u(k-1)-a_2u(k-2)-a_3u(k-3) \end{bmatrix} \qquad (28)$$

donde los coeficiente, debido a la transformación bilineal, quedan expresados en función del período de muestreo. En la Tabla 4 se pueden ver los coeficientes en función del periodo *T*.

Tabla 4. Coeficientes en función de *T*.

Coeficiente	*En Función de T*
a_o	$4{,}9688\cdot T^3+16{,}0938\cdot T^2+18{,}3124\cdot T+8$
a_1	$14.9064\cdot T+16{,}0938\cdot T^2-18{,}3124\cdot T-24$
a_2	$14.9064\cdot T-16{,}0938\cdot T^2-18{,}3124\cdot T+24$
a_3	$4{,}9688\cdot T^3-16{,}0938\cdot T^2+18{,}3124\cdot T-8$
b_0	$T^3+9{,}156\cdot T^2+0{,}4376\cdot T-15{,}7496$
b_1	$3\cdot T^3+9{,}156\cdot T^2-0{,}4376\cdot T+47{,}2488$
b_2	$3\cdot T^3-9{,}156\cdot T^2-0{,}4376\cdot T-47{,}2488$
b_3	$T^3-9{,}156\cdot T^2+0{,}4376\cdot T+15{,}7496$

9.5 Experiencias realizadas.

Para mostrar el comportamiento del mecanismo desarrollado en este capítulo, realizamos simulaciones usando TrueTime. Mostramos las ventajas del mecanismo respecto a un planificador de Prioridades Fijas y al mecanismo desarrollado por Martí en [33]. Se realizaron las siguientes experiencias:

- **Experiencia 1.** Sólo una tarea de control corriendo en el procesador.
- **Experiencia 2.** Una tarea de control junto con tres tareas más que no son de control corriendo en un sistema de tiempo real. El conjunto de tareas fue planificado con una disciplina de prioridades fijas. En tal planificación, a la tarea de control se le asignó la menor prioridad.
- **Experiencia 3.** Idem 2 pero usando el mecanismo propuesto en este capítulo.

- **Experiencia 4.** Idem 2 pero usando el mecanismo propuesto en [33]

En la Tabla 5, se muestran los parámetros de tiempo real del conjunto de tareas usado en las experiencias. Las tareas τ_2, τ_3 y τ_4 fueron introducidas al modelo para modelar el efecto que las tareas que no son de control introducen sobre la aplicación.

Tabla 5. Atributos de las tareas

	τ_1	τ_2	τ_3	τ_4
T	80ms	160ms	240ms	320ms
C	64ms	40ms	45ms	68ms

La Figura 31 muestra la respuesta al escalón de la simulación para cada mecanismo. En esta figura podemos observar que el mecanismo propuesto reduce los efectos adversos de la sobrecarga en un sistema de tiempo real puro. Por otro lado, podemos notar que el sistema alcanza el equilibrio mucho antes que si se usa la compensación desarrollada por Martí.

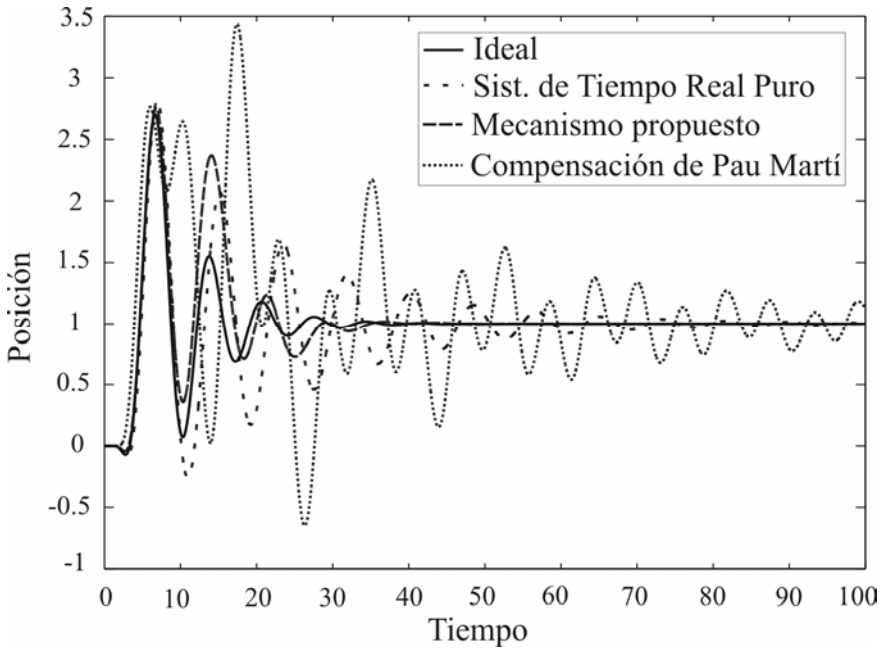

Figura 31**.** Respuesta al escalón para los diferentes casos.

9.6 Análisis y resultados

El mecanismo propuesto alcanza una alta flexibilidad de tiempo real al mismo tiempo que mantiene la eficiencia de control. Trabajos previos sobre este tema

muestran una alta eficiencia de tiempo real, pero ellos pueden no ser convenientes para estrategias de control complejas. Estos se basan en el hecho de que las estrategias complejas del control podrían almacenar una gran cantidad de muestras de la entrada y por lo tanto una modificación brusca del período de muestreo podría conducir a un comportamiento inesperado. El mecanismo propuesto modifica gradualmente el período de las tareas de control y consecuentemente el error introducido es acotado incluso en estrategias de control complejas.

Por otro lado, el mecanismo no fuerza a las tareas de control a tener la mayor prioridad como sucede en otros mecanismos propuestos en la literatura de tiempo real ([16], [4]). Muchas de las aplicaciones requieren la ejecución de tareas de seguridad, las cuales deben ser ejecutadas a los mayores niveles de prioridad para evitar un evento catastrófico. Tales tareas deben ser ejecutadas con un estricto temporizado y sin chances de que el período sea modificado.

Desde el punto de vista de tiempo real, el mecanismo propuesto en este capítulo es bastante flexible para ejecutar adecuadamente una tarea de control en un nivel más bajo de prioridad que otras tareas que no son de control dentro de un sistema de tiempo real.

9.7 Conclusiones

En este capítulo, se propuso un mecanismo para adaptar la carga que las tareas de control producen sobre un sistema de tiempo real. El mecanismo permite alcanzar un alto desempeño de tiempo real sin degradar el desempeño de control. Por otra parte, éste es adecuado para estrategias de control complejas donde el número de muestras pasadas es elevado y el error introducido al cambiar el período de muestreo puede conducir a un comportamiento incontrolado de la aplicación.

Por medio de simulaciones, se pudo verificar que el mecanismo propuesto en este capítulo mejora el propuesto en [33].

Capítulo 10

Conclusiones y Futuros Trabajos

Los sistemas de tiempo real son utilizados en aplicaciones críticas tales como sistemas espaciales, aviación, control de centrales, plantas industriales, donde está en juego no sólo las economías de las empresas, sino algo mucho más importante como lo son las vidas humanas.

El hecho de que el procesador atienda una tarea de control crítica con retraso, podría causar pérdidas catastróficas. Es por eso que es muy importante el estudio y la investigación de esta temática, para encontrar mecanismos de planificación más flexibles de tiempo real para aplicaciones de propósito dedicado en sistemas de control y desarrollar plataformas reales específicamente diseñadas para tiempo real para las mediciones y posterior implementación de éstos.

Bajo esta premisa, en ésta tesis se han analizado y desarrollado un par de mecanismos de planificación en tiempo real de bajo jitter para aplicaciones de propósito dedicado en sistemas de control y se ha presentado una plataforma real basada en técnicas de HW/SW co-design sobre dispositivos lógicos programables (FPGA) para las mediciones e implementación de los algoritmos propuestos.

En la literatura se pueden encontrar muchos trabajos que desarrollan criterios de planificación flexibles de tiempo real para mejorar diferentes características de tiempo real en tiempo de ejecución, pero no exponen resultados sobre las repercusiones que éstos tienen sobre la aplicación de control en sí misma.

Tanto el sistema de tiempo real como la aplicación de control deben ser analizados en forma conjunta para lograr el mejor diseño.

Con el análisis de las perturbaciones que los sistemas de tiempo real producen sobre las aplicaciones de control realizado, se comprobó que éstas pueden tornarlas inestables e incontrolables. De aquí surge la necesidad de encontrar métodos de planificación flexibles.

Uno de los métodos flexibles desarrollados en esta tesis es un mecanismo de prioridades que reduce los efectos adversos que la planificación de tiempo real produce sobre aplicaciones de control. Éste es basado en la promoción de tareas de control a la banda de prioridad superior de acuerdo a la sensibilidad que tales tareas presentan al jitter en tiempo de ejecución.

Los resultados de la comparación del desempeño de PBJ con PF y EDF arrojados de simulaciones usando TrueTime, muestran que PBJ mejora el desempeño de control, y al mismo tiempo, el tiempo de respuesta de las tareas que no son de control. Por otro lado se pudo mostrar que el método flexible tiene efectos más controladores que los clásicos planificadores de tiempo real.

El otro método flexible desarrollado en esta tesis es un mecanismo para adaptar la carga que las tareas de control producen sobre un sistema de tiempo real. El mecanismo permite alcanzar un alto desempeño de tiempo real sin degradar el desempeño de control.

Por medio de simulaciones usando TrueTime, se pudo verificar que el mecanismo propuesto mejora, el presentado por Martí en [33].

Por último, se ha presentado en esta tesis las principales características del procesador uRT51. Se demostró por medio de un ejemplo, que el procesador uRT51 permite planificar un sistema de tiempo real a mucha menos frecuencia que la que necesita un RTOS.

Con el uso del procesador uRT51, el jitter de las tareas de control es reducido, debido a esto, las propiedades de control del sistema de tiempo real son

mejoradas. Por lo tanto, las perturbaciones que el sistema de tiempo real produce sobre la aplicación de control son reducidas y puede alcanzarse una mayor precisión en las características temporales del sistema.

Podemos concluir en que debido a que la arquitectura del procesador uRT51 es adecuada para sistemas de propósito dedicado de tiempo real en aplicaciones de control, los resultados obtenidos mediante los dos mecanismos propuestos en esta tesis podrán ser realmente logrados, mediante una implementación basada en ésta arquitectura.

Queda para futuros trabajos ésta implementación, con la idea de obtener un dispositivo completo basado en una arquitectura de procesamiento específicamente diseñada para tiempo real, que disminuya los efectos adversos que los parámetros de tiempo real producen sobre las aplicaciones de control, adecuado para aplicaciones críticas de control.

Apéndice A

Símbolos

$\Pi(m)$	Conjunto de m tareas de tiempo real.
τ_i	i-ésima tarea de un conjunto de tareas independientes.
T_i	Período de Muestreo Mínimo (de la i-ésima tarea periódica) o mínimo tiempo entre arribo (de la i-ésima y la $i+1$-ésima tarea no periódica).
D_i	Vencimiento de la i-ésima tarea.
C_i	Máximo tiempo de ejecución de la tarea τ_i.
U_i	Factor de utilización de una tarea τ_i.
U_p	Cota superior para analizar la planificabilidad de algoritmos de prioridades fijas.
U	Factor de utilización total del sistema.
w_s	Frecuencia de muestreo ($T=1/w_s$).
w_b	Ancho de banda del sistema a lazo cerrado
t	Variable Tiempo de un sistema continuo
k	Variable Tiempo de un sistema discreto
$\dot{x}$	Operador derivada.
x(t)	Variable de estados
u(t)	Variable de entrada o señal de control
y(t)	Variable de salida, señal de salida del sistema o señal realimentada
f()	Función de estados
g()	Función de estados
e(t)	Señal error
r(t)	Señal de referencia de entrada

A	Matriz del sistema (modelo de ecuaciones de estado de un sist. cont.)
B	Matriz de entrada (modelo de ecuaciones de estado de un sist. cont.)
c	Matriz de salida
d	Matriz de realimentación
Φ	Matriz del sistema (modelo de ecuaciones de estado de un sist. disc.)
Γ	Matriz del entrada (modelo de ecuaciones de estado de un sist. disc.)
$D << T$	D es mucho mayor que T
α	Angulo a partir de la posición vertical del péndulo
$\frac{\partial u[k]}{\partial p_i}$	Operador derivada parcial del la señal $u[k]$ respecto al parámetro p_i
Δ	Operador delta o variación
$\widehat{S}$	Operador máxima sensibilidad
ε	Parámetro precisión de un instrumento
η	Parámetro rango de operación de un instrumento
λ	Constante de proporcionalidad
FP	Disciplina de Prioridades Fijas (en inglés Fixed Priority).
EDF	Disciplina de Prioridades por menor tiempo al vencimiento (en inglés Early Deadline First).
PBJ	Planificado de Bajo Jitter (en inglés Jitter Aware Scheduler)

Apéndice B

TrueTime

B.1 Introducción.

TrueTime es un simulador basado en Matlab/Simulink para sistemas de control de tiempo real desarrollado en [24]. TrueTime permite simular el comportamiento temporal de kernels multi-tareas de tiempo real que contienen tareas de control para estudiar los efectos de planificación de CPU y redes sobre aplicaciones de control.

El kernel de tiempo real simulado es basado en eventos y puede manipular interrupciones externas. Pueden ser definidas diferentes políticas de planificación basadas en vencimientos o basadas en prioridades. Las tareas de control pueden ser implementadas usando funciones programadas en C, o en Matlab o diagramas de bloques de Simulink. El tiempo de ejecución de las tareas de control puede ser modelado de dos maneras, constante o variante en el tiempo usando distribuciones de probabilidad. Son tenidos en cuenta efectos tales como cambio de contexto y manipulación de interrupciones, así como tareas de sincronización usando eventos y monitores. Con TrueTime también es posible simular el comportamiento

temporal de redes de comunicación usadas, por ejemplo, en redes de lazos de control.

TrueTime puede ser usado para varios propósitos: para investigar los efectos reales del temporizado no-determinístico sobre aplicaciones de control, para desarrollar esquemas de compensación que ajustan la dinámica del controlador basándose en medidas de las variaciones de tiempo actuales, para experimentar con planificadores flexibles y dinámicos y para simular sistemas de control basados en eventos.

B.2 El Simulador.

El entorno de simulación de TrueTime ofrece dos bloques Simulink: el bloque de computadora y el bloque de red, las interfaces de tales bloques son mostradas en la Figura 32.

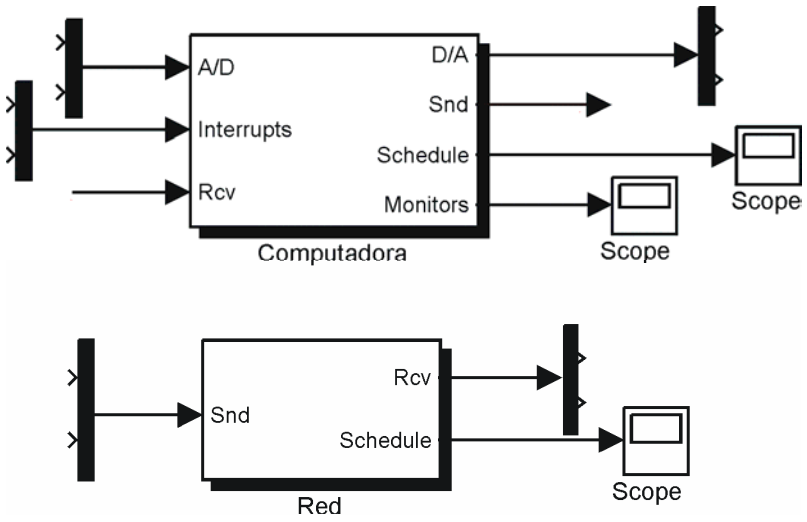

Figura 32**.** Interfaces de los bloques Simulink. Los puertos Schedule y Monitors proporcionar gráficos de la asignación de los recursos comunes (CPU, monitores, red) durante la simulación.

Se asumen que las señales de entrada son discretas, excepto las señales conectadas al puerto A/D las cuales pueden ser continuas. Todas las señales de salida son discretas. Los puertos Schedule y Monitors proporcionan gráficos de la asignación de los recursos comunes (CPU, monitores, red) durante la simulación.

B.2.1 El Bloque de Computadora.

El bloque computadora simula una computadora con un kernel de tiempo real flexible ejecutando threads definidos por el usuario y manipuladores de interrupciones. Los threads pueden ser periódicos o aperiódicos y son usados para simular tareas del controlador, tareas de comunicación, entre otras. El manipulador de interrupciones es usado para servir interrupciones internas y externas. El kernel mantiene las estructuras de datos comúnmente encontradas en kernels de tiempo real, incluyendo una cola de listas, una cola de tiempo, e historiales para threads, manipuladores de interrupciones, eventos, monitores, etc.

El código ejecutado durante la simulación consiste de funciones escritas por usuarios, las cuales pueden ser asociadas con threads y manipuladores de interrupciones. Estas funciones pueden ser escritas en C o en código Matlab.

La ejecución ocurre en tres niveles de prioridad diferentes: nivel interrupción (el mayor), nivel kernel y nivel thread (el menor). La ejecución puede ser apropiativa o no apropiativa. En el nivel interrupción, los manipuladores de interrupciones son planificados de acuerdo a prioridades fijas, mientras que en el nivel thread puede usarse planificación de prioridades dinámicas. Las prioridades de threads son determinadas por funciones de prioridad definidas por el usuario.

Cada thread es definido por un conjunto de atributos, muchos de los cuales son inicializados por el usuario cuando el thread es creado. Estos atributos incluyen: un nombre, un tiempo de activación, vencimientos absolutos y relativos, un tiempo de ejecución, un período (si el thread es periódico), una prioridad (si es planificado usando prioridades fijas), y el código usuario asociado al thread. Algunos de estos atributos, tales como el tiempo de ejecución y el tiempo de ejecución son constantemente actualizados por el kernel durante el tiempo de simulación. Los otros atributos pueden ser cambiados a través del código programado por el usuario.

Cuando ocurre una interrupción interna o externa, el correspondiente manipulador de interrupciones es activado y planificado por el kernel. De manera similar a threads, los manipuladores de interrupciones tienen un conjunto básico de atributos: nombre, prioridad y el código usuario asociado.

La ejecución del código usuario asociado a threads y manipuladores de interrupciones es dividido en segmentos con diferentes tiempos de ejecución simulados como muestra la figura 33 Los tiempos de ejecución pueden ser constantes, aleatorios o dependientes de datos.

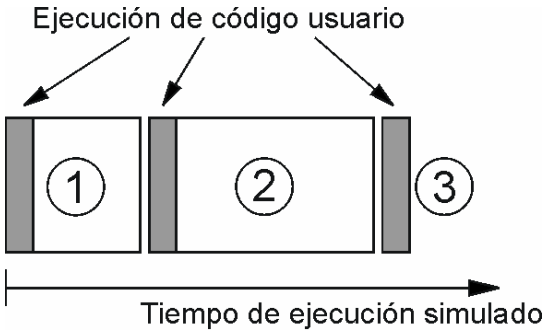

Figura 33**.** La ejecución del código asociado con threads y manipuladores de interrupciones es modelado por un número de segmentos de código con diferentes tiempos de ejecución.

La ejecución del código usuario ocurre al comienzo de cada segmento de código. El siguiente segmento no se ejecuta hasta que el tiempo asociado con el segmento anterior ha transcurrido en la simulación. De esta manera, es posible modelar los aspectos temporales del código que son relevantes para la interacción con otras tareas. Esto incluye, por ejemplo, computación, acciones de entrada y salida, eventos de espera y ejecución en regiones críticas usando monitores. Después de la ejecución del último segmento, es activado el manipulador de finalización de código del thread. Para threads periódicos éste simplemente actualiza la activación y el vencimiento y pone en espera al thread hasta el siguiente período. La ejecución otra vez comenzará en el primer segmento.

B.2.2 El Bloque de Red.

Este bloque es impulsado por eventos y se ejecuta cuando un mensaje entra o sale de la red. Es usada una cola de envío para mantener todos los mensajes actuales en la red. Un mensaje contiene información acerca del nodo computadora que envía el mensaje y del que lo recibe, del dato (señales de medida, señales de control), tiempo de transmisión y atributos de tiempo real tales como prioridad y vencimiento.

B.2.3 Inicialización.

Antes del comienzo de una simulación, los bloques de computadora y de red deben ser inicializados. Esto es hecho en un segmento de código para cada bloque. La inicialización envuelve especificar el número de los puertos de entrada y de salida, la elección de funciones de prioridad, la definición de código de funciones, la creación de threads, manipuladores de interrupciones, etc.

La función código toma como argumento de entrada el segmento a ser ejecutado y retorna el tiempo de ejecución de este segmento. El kernel provee un conjunto de primitivas de tiempo real que pueden ser llamadas desde el código usuario. En la figura 34 se muestran las primitivas más comunes.

`ttanalogout(ch,value)`	`ttanalogin(ch)`
`ttwaituntil(time)`	`ttcurrenttime()`
`ttsetpriority(prio)`	`ttsetrelease(time)`
`ttnwsendmsg(msg,node)`	`ttnwgetmsg()`
`ttentermonitor(mon)`	`ttexitmonitor(mon)`
`ttawait(event)`	`ttcause(event)`

Figura 34**.** Primitivas más comunes de TrueTime.

Una función código para un simple controlador puede ser como la mostrada en la figura 35.

```
function exectime = myController(seg)
switch (seg),
    case 1,
          y = ttAnalogin(1);
          u = calculateOutput(y)
          exectime = 0.002 % execution time
    case 2,
          ttAnalogOut(1,u):
          updateState(y)
          exectime = 0.003 % execution time
    case 3,
          exectime = -1: % code termination
 end
```

Figura 35**.** Función código de un controlador simple.

La latencia de entrada-salida según la figura anterior, es siempre menor que 2 mseg, siendo ésta el tiempo de ejecución del primer segmento. No obstante, la apropiación por threads de mayor prioridad o interrupciones pueden conducir a retardos mayores.

B.3 Conclusiones.

TrueTime es un simulador basado en eventos para co-diseño de sistemas de control y tiempo real. Las simulaciones capturan el comportamiento temporal real de tareas de control de tiempo real. Desde la perspectiva de eficiencia de control, pueden evaluarse la dinámica de control y estrategias de planificación.

Referencias

[1] Adomat, J., Furuns, J., Lindh, L. and Strner, J. “RealTime Kernel in Hardware RTU: A Step Towards Deterministic and High-Performance Real-Time Systems,” In *proc. 8th Euromicro Workshop on Real Time Systems*, L'Aquila, Italy, 164-168, (1996).

[2] Albertos, P., Crespo, A., Ripoll, I., Vallés, M. and Balbastre, P. “RT control scheduling to reduce control performance degrading”. In *proc. of the 39th IEEE Conference on Decision and Control*, (2000).

[3] Astrom, K. J. and Furuta, K. “Swinging Up a Pendulum by Energy Control”. In *proc. 13th IFAC World Congress*, San Francisco, USA, (1996).

[4] Balbastre, P., Ripoll, I., Vidal, J. and Crespo, A. “A Task Model to Reduce Control Delays”. *Journal of Real-Time Systems*, 27, 215-236, (2004).

[5**]** Bernat, G. “Specification and Analysis of Weakly Hard real-Time Systems”. PhD thesis, Departament de Ciències Matemàtiques I Informàtica. Universitat de les Illes Balears. Spain. http://www.cs.york.ac.uk/~bernat, (1998).

[6**]** Bernat, G. and Burns, A. “Jorvik: A framework for effective scheduling”. Technical report (to appear), Department of Computer Science, University of York, (2001).

[7**]** Bernat, G. and Burns, A. "Weakly-hard temporal constraints". Technical report YCS, Department of Computer Science, University of York, (2000).

[8**]** Bernat, G., Burns, A. and Llamosí, A. "Efficient transient overload tests for real-time systems". In *TOOLS-97. Intl. Conference on computer performance evaluation.* St. Malo. France, (1997).

[9**]** Bernat, G., Burns, A. and Llamossí, A. "Weakly Hard Real Time Systems". *IEEE Transactions on Computers,* 50(4): 308-321, (2001).

[10] Bernat, G. and Cayssials, R. "Guaranteed on-line weakly-hard real-time systems". In *proc. 22nd IEEE Real-Time Systems Symposium,* pages 25-35, London, (2001).

[11] Buttazzo, G. and Abeni, L. "Adaptive workload management through elastic scheduling". *Real-Time Systems Journal*, 23:7-24, (2002).

[12] Buttazzo, G. and Caccamo M. "Minimizing Aperiodic Response Times in a Firm Real-Time Environment". *IEEE Transactions on Software Engineering,* 25(1):22-32, (1999).

[13] Caccammo, M. and Buttazzo, G. "Exploiting skips in periodic tasks fo enhancing aperiodic responsiveness". In *proc. 17th IEEE Real-Time Systems Symposium,* pages330-339, San Francisco, CA, (1997).

[14] Caccamo, M., Buttazzo, G. and Sha, L. "Elastic Feedback Control". In *proc. 12th IEEE Euromicro Conference on Real-Time Systems*, pp.:121-128, Sweden, (2000).

[15] Cervin, A. "Integrated Control and Real-Time Scheduling". PhD thesis, Department of Automatic Control. Lund Institute of Technology, (2003).

[16] Cervin, A. and Eker, J. "The Control Server: A Computational Model for Real-Time Control Tasks". In *proc. 15th Euromicro Conference on Real-Time Systems*, Portugal, (2003).

[17] Davis, R. and Wellings, A. "Dual Priority Scheduling". Real-Time Systems Symposium, pp. 100-109, (1995).

[18] Glaviníc, V., Gros, S. and Colnaric, M. "Vhdl-based modelling of a hard real-time task processor". In *proc. 1999 IEEE International Symposium on Industrial Electronics*, Bled, Slovenia, 49-54, (1999).

[19] Halang, W. and Colnaric, M. "Architectural support for predictability in hard real time systems". *Control Engineering Practice*, **1**(1), 281–285 (1993).

[20] Halang, W., Colnaric, M. and Verber, D. “Supporting high integrity and behavioural predictability of hard real-time systems”. Informatics, Special Issue on Parallel and Distributed Real-Time Systems, 19(1), 59–69 (1995).

[21] Hamdaoui, M. and Ramanathan, P. “A dynamic priority assignment technique for streams with (*m*, *k*)-firm deadlines”. *IEEE Transactions on Computers,* 44(12):1443-1451, (1995).

[22] Hamdaoui, M. and Ramanathan, P. “Evaluating dynamic failure probability for streams with (m, k)-firm deadlines”. *IEEE Transactions on Computers*, 46(12):1325-1337, (1997).

[23] Henriksson, D and Cervin, A. “Optimal On-line Sampling Period Assignment for Real-Time Control Tasks Based on Plant State Information”. In *proc. 44th IEEE Conference on Decision and Control and European Control Conference ECC 2005*, Spain, (2005).

[24] Henriksson, D., Cervin, A., Åkesson, J. and Årzén, K-E. “Feedback scheduling of model predictive controllers”. In *proc. of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium*, IEEE-Computer Society Press. San Jose, CA, (2002).

[25] Henriksson, D., Cervin, A. and Årzén, K-E. “TrueTime: Simulation of Control Loops Under Shared Computer Resources”. In *Proc. of the 15th IFAC World Congress on Automatic Control*, Barcelona, Spain, (2002).

[26] Henrion, D., Hansson, A. and Wallin, R. “Reduced LMIs for fixed-order polynomial controller design”. In *proc. 16th International Symposium on Mathematical Theory of Networks and Systems*. pp. 1-12, Belgium, (2004).

[27] Hildebrand, D. “An Architectural Overview of QNX”. In *proc. 1992 Usenix Workshop on Micro-Kernels & Other Kernel Architectures*, Seattle, USA, 113-126, (1992).

[28] IEEE1003.1d. Draft Information Technology - Portable Operating System Interface (POSIX): Part 1: System Application program interface; Additional Real-Time Extensions, IEEE Standards, (1999).

[29] Koren, G. and Shasha, D. “Skip-over: Algorithms and complexity for overloaded systems that allow skips”. In *proc. 16th IEEE Real-Time Systems Symposium,* pages 110-117, Pisa, Italy, (1995).

[30] Liu, C. L., and Layland, J. W. “Scheduling algorithms for multiprogramming in hard real time environments”. *J. ACM*, **20**(1): pp. 46-61. (1973).

[31] Mackall, D. A., "Development and flight test experiences with a flight-crucial digital control system". Internal Report, Technical Paper 2857, NASA AMES Research Center, NASA, (1988).

[32] Martí, P., Fuertes, J. M., Fohler, G. and Ramamrithan, K.: "Improving quality-of-control using flexible timing constraints: Metrics and scheduling issues". In *proc. 23rd IEEE Real-Time Systems Symposium,* pages 91-100, (2002).

[33] Martí, P., Fuertes, J. M., Fohler, G. and Ramamrithan, K. "Jitter compensation for real-time control systems". In *proc. 22nd IEEE Real-Time Systems Symposium*, pages 39-48, London, (2001).

[34] Matthew, M., S. Wilding, P. Miller, D. A. Greve and M. Srivas., "Formal verification of the AAMP-FV microcode," *Internal Report MD21076-1320*, NASA, (1999)

[35] Ordinez, L., Donari, D., Sanchez, D y Duval, M. "A Semantic-Based Scheduling Approach for Soft Real-Time Systems". In *proc. XXXIII Conferencia Latinoamericana de Informática, CLEI 2007*, 9-12, San José, Costa Rica, (2007).

[36] Quan, G. and Hu, X. Enhanced fixed-priority scheduling with (m, k)-firm guarantee. In *12th IEEE Euromicro Conference on Real-Time Systems*, Sweden, (2000).

[37] Seto, D., Lehoczky, J. P., Sha, L. and Shin, K. G. "On Task Schedulability in Real-Time Control System". In *proc. of the IEEE RTSS*, (1996).

[38] Sha, L., Abdelzaher, T., Årzén, K-E., Cervin, A., Baker, T., Burns, A., Caccamo, M., Lehoczky, J. and Mok, A. K. "Real Time Scheduling Theory: A Historical Perspective". *Real-Time Systems* 28(2-3):101-155, (2004).

[39] Shiriaev, A. S., Friesel, A., Perram, J. And Pogromsky, A. "On Stabilization of Rotational Models of an Inverted Pendulum". In *proc. of th 39th Conference on Decision and Control*, (2000).

[40] Sitio Web del uRT51, http://www.uRT51.com.ar, (2007).

[41] Stankovic, J. A. and K. Ramamritham, "The Spring Kernel: A New Paradigm for Real-Time Systems". IEEE Software, 8(3), 62-72, (1991).

[42] Törngren, M. "Fundamentals of implementing real-time control applications in distributed computer systems". *Journal of Real-Time Systems*. Control and Real-Time scheduling, 14:219-250, (1998)

[43] Vaccaro, R. J., "Digital Control: A State-Space Approach". McGraw-Hill Series in Electrical and Computer Engineering, (1995).

[44] West, R. and Poellabauer, C. "Analysis of a window-constrained scheduler for real-time and best-effort packet streams". In *21st IEEE Real-Time Systems Symposium*. Florida. USA, (2000).

[45] West, R., Schwan, K. and Poellabauer, C. "Scalable scheduling support for loss and delay constrained media streams". In *5th IEEE Real-Time Technology and Applications Symposium*. Vancouver, (1999).

[46] Wie, B. and Bernstein, D. S. "Benchmark problems for robust control design", *AIAA Journal of Guidance, Control and Dynamics*, Vol. 15, No. 5, pp. 1057-1059, (1992).

[47] Wittenmark, B. and Trngren, N.: Timing problems in real-time control systems. In *proc. 1995 American Control Conference*, (1995).