

INSTITUTO TECNOLÓGICO DE COSTA RICA
ESCUELA DE INGENIERÍA ELECTRÓNICA

Informe de Proyecto de graduación para optar por el título de Ingeniería
Electrónica con el grado académico de Licenciatura

Plataforma de Automatización para un Sistema Robótico Utilizando Software Libre y Abierto

Carlos José Corrales Chinchilla

2011

INSTITUTO TECNOLOGICO DE COSTA RICA

ESCUELA DE INGENIERIA ELECTRONICA

PROYECTO DE GRADUACIÓN

TRIBUNAL EVALUADOR

ACTA DE EVALUACIÓN

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Estudiante: Carlos Corrales Chinchilla

Carné: 200409109

Nombre del Proyecto: Plataforma de Automatización para Sistemas Robóticos
Utilizando Software Libre y Abierto

Miembros del Tribunal



Ing. Marvin Hernández.

Profesor lector



Ph.D. Ing. Carlos Meza Benavides

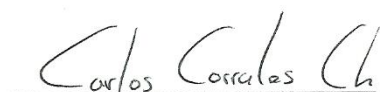
Profesor asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica

Cartago, Costa Rica
Lunes 25 de junio del 2012

Yo Carlos José Corrales Chinchilla garantizo que la totalidad del material ilustrativo presentado en este documento fue generado por mi persona, al igual que todas las ideas que en él se plantean referentes a la formulación del proyecto.

Además hago constar que la información suministrada fue recopilada respetando la autoría de sus respectivas fuentes.

A handwritten signature in black ink that reads "Carlos Corrales Ch." with a horizontal line underneath.

Carlos Corrales Chinchilla

Cédula 1-1300-0769

Plataforma de Automatización para un Sistema Utilizando Software Libre Y Abierto

Resumen

Los Sistemas Robóticos juegan un papel protagónico en muchas industrias, tales como la automotriz, en organización de inventarios de gran escala, en la manipulación de materiales nocivos para seres humanos, soldadura, ensamblaje, pintura, etc. Todos los trabajos mencionados anteriormente los pueden realizar con un alto rendimiento, precisión y velocidad.

Todo Sistema Robótico requiere un Controlador, que permita la el funcionamiento del mismo, para su correcta operación. En este informe se presenta una solución alternativa a los Controladores producidos por la empresas productoras de Sistemas Robóticos.

La solución basa su funcionamiento en Software libre y Abierto específicamente en RTAI(Real Time Application Interface) compilada con el kernel de Linux, para poder usar una distribución de Linux como un Sistema Operativo de Tiempo Real, además de una tarjeta de adquisición de datos de la empresa National Instruments, controlado por la interfaz Comedi, que ofrece los drivers necesarios para la tarjeta de National instruments.

Automation Platform for Robotic Systems Control Using Free and Open-Source Software

Abstract

Robotic Systems are widely used in several industries, such as: the automotive industry, pick and place, handling materials, welding, painting, etc. All the previous activities are done with high performance, precision and speed.

Every Robotic System needs a Controller, which allows the proper operation of the robot. This paper presents an alternative solution to the Controllers produced by the manufacturer companies of Robotic Systems.

The solution is based on Free and Open-Source Software, using RTAI(Real Time Application Interface) , compiled with the Linux Kernel, so that it can run as a Real Time Operating System, along with a Data Acquisition Board (DAQ) produced by National Instruments, controlled by the Comedi interface, which offers the drivers so that, the DAQ can operate under Linux.

Dedicatoria

A mi familia, que siempre ha estado a mi lado.

A mi Madre, quien ha sido un ejemplo de esfuerzo, dedicación y trabajo duro para siempre seguir mis metas y sueños, aun en los momentos mas adversos.

A mi Padre, por enseñarme a siempre dar lo mejor de mi.

A mi Hermana, quien siempre ha estado a mi lado, apoyándome en los momentos difíciles, y acompañadome en los momentos alegres.

A Caro, por su apoyo incondicional en todo momento.

Agradecimiento

Al profesor Ing. Carlos Meza, por haberme dado la oportunidad de realizar mi proyecto bajo su supervisión.

A todos mis amigos del TEC, ellos hicieron de la etapa universitaria una experiencia amena.

A Eduardo, por haber hecho divertidas las horas de trabajo en SESLab.

Índice

1	Introducción.....	1
1.1	Problema existente e importancia de su solución	1
1.2	Solución Seleccionada	1
2	Meta y Objetivos	3
2.1	Meta.....	3
2.2	Objetivo general	3
2.3	Objetivos Específicos	3
3	Marco Teorico	4
3.1	Control Automatico.....	4
3.2	Sistema Mecánico	4
3.3	Sistema Operativo (OS)	4
3.3.1	Kernel.....	4
3.3.2	Programacion de aplicaciones para un sistema operativo.....	5
3.4	Sistema Operativo de Tiempo Real (RTOS).....	5
3.4.1	Aplicaciones para Tiempo-Real.....	6
3.4.2	Ventajas de un RTOS sober un Sistema Operativo general.....	6
3.5	Linux	7
3.6	RTAI (Real-Time Application Interface).....	7
3.6.1	Fundamentos de RTAI.....	7
3.6.2	LXRT: Interfaz de “User-space” para RTAI	8
3.7	Adquisición de Datos [10].....	8
3.7.1	Señales físicas de Entrada/Salida	9
3.7.2	Dispositivo/Hardware de DAQ	9
3.7.3	Software Controlador (Driver Software).....	9
3.7.4	Programa de Software de Aplicación	9
3.8	Comedi	10
3.8.1	Jerarquía del Dispositivo(tarjeta DAQ).....	10
3.8.2	Funciones DAQ.....	11

3.9	Convertidor DC/DC	11
3.10	PWM.....	11
4	Solución.....	12
4.1	Sistema de Administración y Control (SAC).....	12
4.1.1	Sistema Operativo	12
4.1.2	Paquetes de software requeridos por RTAI.....	12
4.1.3	Requerimeintos de hardware y software:	12
	Software: Utilizar alguna distribución disponible de Linux como Ubuntu.	12
4.1.4	Pasos para la instalación de RTAI	12
4.1.5	Programación en C para el uso de la tarjeta DAQ.....	13
4.1.6	Programacion de una tarea de tiempo real en <i>user-space</i>	14
4.2	Comprobación del SAC por medio de un Sistema Mecánico.....	15
4.3	Esquema General del Control del Sistema Mecánico	16
4.3.1	Sistema de Control	16
4.3.2	Planta (G_p).....	16
4.3.3	Controlador (G_c).....	16
4.3.4	Etapas de Potencia	17
4.3.5	PWM	17
4.3.6	Convertidor Buck	19
4.3.7	Elección de los componentes del Convertidor Buck.....	20
4.3.8	Sensor de posición.....	22
4.4	Esquema general del Sistema de Control.....	23
4.5	Modelado Matemático	23
4.6	Caracterización De la Planta	25
4.7	Diseño del Controlador de Posición.....	29
4.7.1	Respuesta en Régimen Permanente	30
4.7.2	Estabilidad del Sistema de Control de Posición.....	30
4.7.3	Respuesta transitoria	31
4.7.4	Implementación del controlador en el SAC	32

5	Resultados.....	34
6	Análisis de resultados	39
7	Conclusiones.....	41
8	Recomendaciones	42
	Bibliografía	43
	Anexos.....	45
	Anexo1: Instalación de RTAI en Ubuntu 10.04 LTS	45
	Anexo 2: Modelado Matemático del Buck y la planta.....	55
	Anexo 3: Controlador implementado utilizando las capacidades de RTAI.....	56

1 Introducción

1.1 Problema existente e importancia de su solución

Las empresas manufactureras de Sistemas Robóticos se encargan de producir tanto los Robots, como Controladores, necesarios e indispensables para la operación y programación de las rutinas o labores que se quiere, sean realizadas por el Sistema Robótico.

Debido al alto costo de los Controladores, e Interfaces comerciales, se hace necesario el desarrollo de una solución alternativa, de menor precio que los comerciales.

A la hora de adquirir un Sistema Robótico se debe adquirir por separado el Controlador, componente que es casi tan caro como el robot mismo, lo que hace que la inversión por parte del interesado en el Sistema Robótico, sea considerable.

Los Sistemas Robóticos pueden ser adquiridos de segunda mano (usados), a un menor precio, al igual que el controlador, sin embargo si se desea abaratar los costos aun mas, se puede prescindir de la compra del controlador, producido por la empresa manufacturera.

Un Sistema Robótico sin el controlador, es prácticamente inútil, por lo que surge la idea de desarrollar uno de bajo costo, para poder usar el Robot con todas sus capacidades.

Los controladores comerciales basan su funcionamiento en un Sistema Operativo de Tiempo Real. Las versiones comerciales de este tipo de Sistemas Operativos son sumamente costosas, por lo que surge la idea de desarrollar una Plataforma de Control basado en un Sistema Operativo de Tiempo Real basado en Software Libre y Abierto.

1.2 Solución Seleccionada

En los últimos años las computadoras personales se han convertido en la plataforma mas ampliamente usada para la adquisición de datos y control. Los principales motivos de la popularidad de la tecnología basada en Computadoreas Personales, son:

- Bajo costo.
- Flexibilidad.
- Facilidad de uso.
- Alto Desempeño.

La adquisición de datos con una Computadora Personal posibilita mostrar, registrar y controlar una amplia variedad de señales del mundo real como presión, voltaje, corriente, etc. [1]

En el área de Ingeniería de Control, es necesario un entorno computacional que facilite el diseño, la simulación, la implementación de algoritmos de control, y por último que permita interactuar con la planta real a controlar.

Existen productos en el mercado que incluyen tanto el software como el hardware necesarios que proporcionan las funcionalidades mencionadas anteriormente, pero son altamente costosos. Tal es el caso de Matlab/Simulink/Real-Time Workshop, que junto con tarjetas de adquisición de datos simplifican grandemente el proceso de experimentación.

La contraparte de los productos comerciales costosos la ofrece la comunidad Open Source que ofrece una lista grande de herramientas de software que pueden ser usadas, distribuidas, y modificadas bajo la licencia GPL¹.

En el caso del área de Ingeniería de Control la comunidad Open Source ofrece los proyectos RTAI² que permite implementar aplicaciones de Tiempo Real en Linux; y Comedi³ que está relacionado con la utilización de Tarjetas de Adquisición de Datos.

Por lo tanto *Linux*, *RTAI* y *Comedi*, han sido seleccionados para el desarrollo de la *Plataforma de Administración y Control*, ya que ofrecen todas las capacidades para implementar un sistema de control y además se tienen todas las demás ventajas ofrecidas por una computadora.

¹ La General Public License (GPL) es la licencia de software libre mas ampliamente usada.

² Real Time Application Interface para Linux, permite escribir aplicaciones(de software) con restricciones estrictas de tiempo.

³ Comedi es la interfaz entre dispositivos de control y medición y Linux.

2 Meta y Objetivos

2.1 Meta

Obtener una Plataforma de Automatización para Sistemas Robóticos basada en Software Libre y Abierto.

2.2 Objetivo general

Diseñar e implementar un Sistema Prototipo de Control, para un Sistema Robótico, basado en Software Libre y Abierto.

2.3 Objetivos Específicos

1. Desarrollar un Sistema de Medición y Actuación en Tiempo Real (SMA-TR), corriendo en una computadora personal.
2. Diseñar e implementar un Circuito Acondicionador de Potencia (CAP), que permita controlar una planta física para la comprobación del (SMA-TR).
3. Diseñar e implementar un Sistema de Administración y Control (SAC), que integre el SMA-TR y el CAP, que permita al Sistema Robótico ubicarse en un punto previamente definido.

3 Marco Teorico

3.1 Control Automatico

Un sistema de Control Automático es una interconexión de componentes que forman una configuración del sistema que proporciona una respuesta desada [2].

3.2 Sistema Mecánico

Un Sistema Mecánico es un dispositivo, multifuncional, programable, diseñado para mover materiales, partes, herramientas, o para realizar diferentes tareas [3].

3.3 Sistema Operativo (OS)

Es un programa que funciona como intermediario entre el usuario de una computadora y el hardware de la computadora. El propósito de un Sistema Operativo es el de proveer un ambiente en el cual el usuario puede ejecutar programas de manera eficiente y conveniente.

Un Sistema Operativo tiene distintos componentes, que existen con el propósito de hacer que las diferentes partes de una computadora funcionen juntas. Cualquier software, necesita pasar a través del sistema operativo con el propósito de usar el hardware necesario para que funcione, así sea algo simple como un mouse o teclado o algo tan complejo como una conexión a Internet. El Kernel es el principal componente de un Sistema Operativo. [4]

3.3.1 Kernel

Es el principal componente de la mayoría de los sistemas operativos, es un “puente” entre las aplicaciones (software) y el verdadero procesamiento de información que se da a nivel de hardware. La principal responsabilidad del kernel es la de administrar los recursos de hardware del sistema (la comunicación entre hardware y software).

Los principales recursos de hardware que administra el kernel son:

- **Central Processing Unit (CPU):** Esta es la parte más importante de una computadora, es responsable de ejecutar los programas.
- **Memoria:** la memoria se usa principalmente para almacenar tanto instrucciones como información.
- **Dispositivos de Entrada/Salida (I/O Devices)** que estén presentes en la computadora como teclado, mouse, displays, Data Acquisition Board (DAQ) etc.

El kernel asigna solicitudes hechas por las aplicaciones para ejecutar operaciones de Entrada/Salida al dispositivo (o subsección del dispositivo) adecuado. [5]

La interacción entre las aplicaciones, el kernel y el hardware de una computadora se muestran en la figura 1.

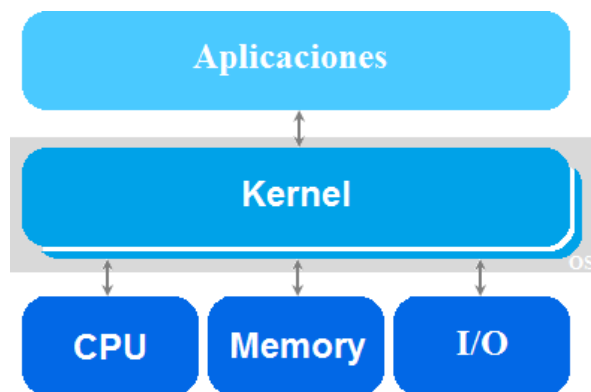


Figura 1. Diagrama de bloques de la interconexión entre el kernel y el hardware de una computadora.

3.3.2 Programación de aplicaciones para un sistema operativo

Solo procesos privilegiados pueden correr en el kernel, donde tienen acceso a todo el hardware y a todas las estructuras de datos. Los programas de aplicaciones normales pueden correr sus procesos en user-space.

3.4 Sistema Operativo de Tiempo Real (RTOS)

Un Sistema Operativo de Tiempo Real, habilita una temporización precisa de múltiples tareas en un orden más estricto que los Sistemas Operativo tradicionales. [6]

El procesamiento en tiempo real puede definirse como un tipo de procesamiento en el que la exactitud del sistema no depende solo del resultado lógico de un cálculo sino también del instante en que se produzca el resultado.

La funcionalidad principal y requerida para un Sistema Operativo de Tiempo Real es proveer de un nivel de servicio adecuado a las aplicaciones que requieran una respuesta en un intervalo de tiempo determinístico.

El criterio fundamental de evaluación del rendimiento de un Sistema Operativo de Tiempo Real es la *Latencia* ante un evento y el periodo de fluctuación. Un evento es cualquier tipo de interrupción, puede ser tanto una interrupción hardware como una interrupción software.

La *Latencia* ante una interrupción de hardware es el tiempo desde que se produce la interrupción hasta que se ejecuta la primera instrucción de la rutina de servicio. La *Latencia* ante una interrupción de software es el tiempo desde que la señal es generada hasta que se ejecuta la primera instrucción de la rutina de servicio.

Si una tarea se debe ejecutar de manera periódica, se denomina fluctuación a la variación del periodo que sufre debido a varios factores relacionados con la ejecución de las instrucciones en el procesador. [7]

En general un Sistema Operativo de Tiempo Real, ejecuta las mismas tareas que un Sistema Operativo de Propósito General (como Windows), pero está diseñado para correr aplicaciones con una temporización muy precisa y un alto grado de confiabilidad.

Para considerar de Tiempo Real a un OS, este debe tener un tiempo máximo **conocido** de ejecución para algunas aplicaciones (o al menos poder garantizar ese máximo la mayoría de las veces). Los RTOSs que pueden garantizar por completo un tiempo máximo para estas aplicaciones son llamados *Sistemas Operativos de Tiempo Real Estricto (hard Real-Time)*, los que pueden garantizar un máximo solamente la mayoría de las veces son llamados *Sistemas Operativos de Tiempo Real flexible (Soft Real-Time)*.

3.4.1 Aplicaciones para Tiempo-Real

Los RTOSs fueron diseñados para dos clases de aplicaciones generales: respuesta de eventos y control a lazo cerrado.

Un sistema de control de lazo cerrado continuamente procesa información retroalimentada para ajustar una o más salidas. Debido a que cada valor de salida depende de procesar la información en un cantidad fija de tiempo, es crítico que las salidas correctas son producidas a tiempo.

3.4.2 Ventajas de un RTOS sobre un Sistema Operativo general.

Prioridad de las aplicaciones

Cuando se programa una aplicación, los sistemas operativos permiten al programador especificar una prioridad. Un sistema operativo de propósito general no siempre siguen estrictamente estas prioridades programadas.

En contraste un RTOS sigue las prioridades establecidas por el programador más estrictamente.

Latencia de las interrupciones

Un RTOS asegura una latencia de un máximo de tiempo para atender las interrupciones, a diferencia de un sistema operativo de propósito general.

Desempeño

Un RTOS puede ofrecer mejor desempeño que uno de propósito general, sin embargo esta característica depende principalmente de la velocidad del CPU, la arquitectura de memoria entre otros. [8]

3.5 Linux

Es un Sistema Operativo para computadoras, basado en Unix⁴, concebido bajo el modelo de desarrollo y distribución de código libre y abierto. El componente determinante de Linux es el *Kernel de Linux*.

3.6 RTAI (Real-Time Application Interface)

Real-Time Application Interface es una implementación de Tiempo Real en Linux basado en RTLinux⁵, la cual habilita a Linux para funcionar como un Sistema Operativo de Tiempo Real.

RTAI agrega un kernel pequeño de tiempo real por debajo del kernel standard de Linux, y trata al Kernel de Linux como a una tarea de tiempo real de baja prioridad.

RTAI provee una larga selección de mecanismos de procesos de intercomunicación y otros servicios de tiempo real. Adicionalmente RTAI provee el módulo LXRT para el desarrollo sencillo de aplicaciones de tiempo real. LXRT hace posible cambiar dinámicamente entre operaciones de tiempo real y de no tiempo real en “User Space”.

3.6.1 Fundamentos de RTAI

Al igual que RTLinux, RTAI trata al kernel de Linux como una tarea de tiempo real de baja prioridad, la cual puede hacer sus operaciones normales siempre y cuando no hay tareas de mas alta prioridad siendo ejecutadas. En la operación básica de RTAI las tareas son implementadas como modulos del Kernel de Linux. RTAI manipula las interrupciones de los

⁴ Unix es un sistema operativo para computadoras, multi-tareas y multi-usuarios.

⁵ Es un microkernel de un Sistema Operativo de Tiempo Real Estricto que corre Linux como un sistema completamente predecible.

periféricos y envía las interrupciones al kernel de Linux luego de haber manipulado las posibles acciones de tiempo real disparadas por las interrupciones.

3.6.2 LXRT: Interfaz de “User-space” para RTAI

LXRT es una API⁶ para RTAI la cual hace posible desarrollar aplicaciones de tiempo real completamente en user-space, sin tener que crear módulos de kernel, lo cual facilita la programación de las tareas de tiempo real. [9]

3.7 Adquisición de Datos [10]

El propósito de la adquisición de datos es medir un fenómeno eléctrico o físico, tal como lo es Voltaje, Corriente, Temperatura, Presión, o Sonido. La Adquisición de Datos basada en PC usa una combinación de hardware modular, programas de software, y una computadora para tomar medidas. Mientras el sistema de adquisición de datos es definido por los requisitos de aplicación, cada sistema comparte un objetivo común, que es **adquirir, analizar, y presentar información**. Los sistemas de adquisición de datos incorporan señales, sensores, actuadores, acondicionamiento de señal, dispositivos de adquisición de datos, y programas de software.

En resumen, la adquisición de datos es el proceso de:

- Adquirir datos de fenómenos del mundo real.
- Digitalizar las señales.
- Analizar, presentar y guardar la información.

Las partes de un sistema DAQ se muestran en la figura 2:

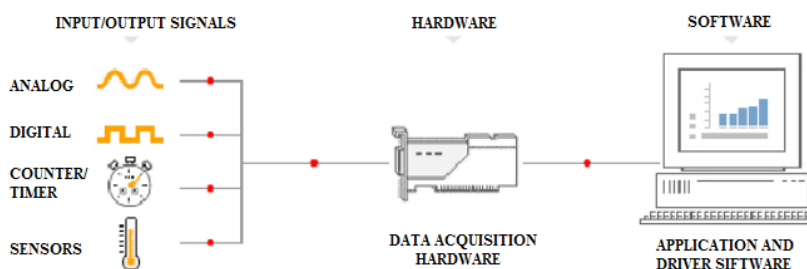


Figura 2. Sistema DAQ.

⁶ Application Programming Interface, es una especificación con el propósito de ser usada como una interfaz de componentes de software para comunicarse entre ellos

Las partes son:

- Señales físicas de Entrada/Salida.
- Dispositivo/Hardware de DAQ.
- Software Controlador (Driver Software).
- Programa de Software de Aplicación

3.7.1 Señales físicas de Entrada/Salida

Es típicamente un Voltaje o una Corriente.

3.7.2 Dispositivo/Hardware de DAQ

Actúa como una interfaz entre la computadora y el mundo exterior. Principalmente funciona como un dispositivo que digitaliza señales analógicas entrantes, para que la computadora las pueda interpretar.

Un dispositivo DAQ (Hardware de Adquisición de Datos) normalmente tiene estas funciones:

- Entrada Analógica.
- Salida Analógica.
- Entrada/Salida Digital.
- Contadores/Temporizadores.

Hay diferentes dispositivos de DAQ, uno de ellos es el de escritorio, esta se utiliza cuando se necesita conectar una tarjeta DAQ PCI a la computadora.

3.7.3 Software Controlador (Driver Software)

El Software Controlador es la capa de software que permite comunicarse fácilmente con el hardware. Forma la capa intermedia entre el Programa de Software (Aplicación) para la aplicación específica y el hardware. Este también evita que el programador tenga que programar a nivel de registros o con comandos complicados, con el propósito de acceder las funciones de hardware.

3.7.4 Programa de Software de Aplicación

Este agrega capacidades de análisis y presentación al Software Controlador. El Programa de Software de Aplicación realiza tareas como:

- Monitoreo en Tiempo Real.

- Analisis de Datos.
- Registro de información.
- Algoritmos de control.
- Interfaz entre Humanos y Máquinas.

3.8 Comedi

Comedi es proyecto de software libre que desarrolla controladores, herramientas y librerías, para varias formas de adquisición de datos:

- Lectura y escritura de señales analógicas.
- Lectura y escritura de entradas/salidas digitales.
- Contadores de pulso y frecuencia.
- Lectura de encoders.
- Entre otras.

Comedi puede trabajar con los kernels estándar de Linux, como también con las extensiones de tiempo real como RTAI.

A continuación se presenta un vistazo general a la forma en que trabajan los Driver de Comedi.

3.8.1 Jerarquía del Dispositivo(tarjeta DAQ)

Comedi organiza todo el hardware de acuerdo a la siguiente jerarquía:

- **Canal:** el componente de hardware de mas bajo nivel. Que representa las propiedades de cada canal de información; por ejemplo, una entrada analógica o una salida analógica.
- **Subdispositivo:** un arreglo de canales funcionalmente idénticos. Por ejemplo un arreglo de 16 entradas analógicas idénticas.
- **Dispositivo:** un arreglo de subdispositivos que son físicamente implementados en la misma tarjeta de interfaz. En otras palabras la tarjeta misma. Por ejemplo la National Instruments 6221, tiene un subdispositivo con 16 canales de entrada analógica, otro con 2 canales de salida analógica, y un tercero con 24 entradas/salidas digitales.

3.8.2 Funciones DAQ

Adquisición simple: comedi tiene funciones de llamada para realizar sincrónicamente la adquisición de un dato, en un canal específico, como:

- comedi_data_read.
- comedi_data_write.

Síncrono significa que el proceso de llamado, se bloquea hasta que la adquisición del dato ha finalizado. [11]

3.9 Convertidor DC/DC

Es el convertidor DC/DC reductor controlado por PWM básico, puede convertir el voltaje de alimentación en voltajes menores, y aseguran una salida “suave” de voltaje DC para alimentar un Motor DC, sin dañar los componentes internos del mismo. [12]

3.10 PWM

Modulación de Ancho de Pulso. Circuito basado en una onda triangular y un comparador, para generar una onda cuadrada con ciclo de trabajo variable, necesaria para controlar la tensión de salida del Buck.

4 Solución

4.1 Sistema de Administración y Control (SAC)

Este sistema se desarrolló en una computadora de escritorio marca Dell, con un procesador Core 2 Duo. Cada una de las partes tanto de Software como de Hardware se presentan en las subsecciones siguientes:

4.1.1 Sistema Operativo

El Sistema Operativo utilizado es Linux, en su distribución Ubuntu 10.04 LTS, este servirá como plataforma para correr los algoritmos de adquisición de datos y de control.

El kernel del Sistema Operativo debe ser “modificado” para que Linux pueda funcionar como un Sistema Operativo de Tiempo Real, para tal fin se utilizó el parche de software RTAI.

4.1.2 Paquetes de software requeridos por RTAI

La instalación completa de RTAI requiere que varios paquetes de software sean descargados y compilados desde su fuente. Estos paquetes son:

- Un kernel de Linux nuevo (Vanilla Kernel) al que se le aplicará parche de código RTAI.
- La librería interfaz de dispositivos de adquisición de datos, Comedilib.
- Los módulos de tiempo real de RTAI.
- Los módulos de los dispositivos de adquisición de datos, Comedi.

4.1.3 Requerimientos de hardware y software:

Hardware: RTAI admite varias arquitecturas: x86, x86_64, PowerPC, ARM, m68k. Una tarjeta madre con puerto PCI para conectar la tarjeta DAQ.

Software: Utilizar alguna distribución disponible de Linux como Ubuntu.

4.1.4 Pasos para la instalación de RTAI

1. Compilar e instalar las librerías *Mesa* y *EFLTK* entre otros paquetes de software.

2. Aplicar el parche *RTAI* al Kernel Vanilla⁷ descargado: una vez aplicado, reiniciar la computadora en el nuevo kernel.
3. Primera instalación de *RTAI*: configurar el kernel para que reconozca el número de procesadores del CPU.
4. Pruebas de *RTAI*: comprobar que *RTAI* ha sido instalado correctamente por medio de la prueba de latencia.
5. Instalacion de *Comedi*.
6. Segunda instalación de *RTAI*: configura *RTAI* para que admita la tarjeta de adquisición de datos.

En este punto al reiniciar la computadora en el kernel de *RTAI* esta debería operar en tiempo real. [13]

4.1.5 Programación en C para el uso de la tarjeta DAQ.

La tarjeta de adquisición de Datos (DAQ) utilizada fue la *NI PCI-6221*, la cual tiene 16 entradas analógicas, 24 entradas/salidas digitales y 2 salidas analógicas, esta tarjeta se comunica con la PC por medio del puerto PCI, la misma se muestra en la figura 2

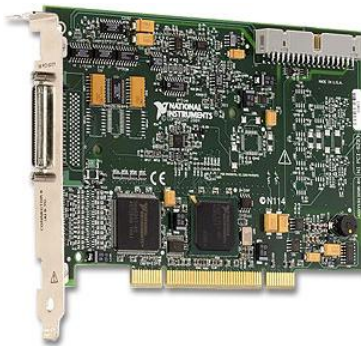


Figura 3 Tarjeta NI PCI-6221.

La programación en C para la adquisición de Datos se realizó siguiendo la guía disponible en la documentación de *comedi.org*. Los principales puntos de este código en C son:

- `it = comedi_open("/dev/comedi0")`: este comando permite abrir el software controlador (Driver) necesario para poder comunicarse con la tarjeta DAQ por medio de las funciones ofrecidas por *Comedi*.

⁷ Kernel que no ha sido modificado previamente.

- *comedi_data_read(it, subdev, chan, range, aref, &data)*: este comando permite realizar una operacion de **adquisición(lectura)** simple, en un subdispositivo especificado por *subdev*, en el canal especificado por *chan*, y almacenar el dato digital en la variable *&data*.
- *comedi_data_write(it, subdev, chan, range, aref, &data;)*: este comando permite realizar una operacion de **escritura** simple, en un subdispositivo especificado por *subdev*, en el canal especificado por *chan*, y almacenar el dato digital en la variable *&data*. [14]

4.1.6 Programacion de una tarea de tiempo real en *user-space*

Una tarea de tiempo real en user-mode es la forma mas simple de correr una tarea deseada en tiempo real. Es una simple tarea de GNU/Linux con sus funciones *main()*; pero al “llamar” algunas API’s especificas de RTAI se convertirá en una tarea de tiempo real corriendo en user-space. Para poder correr un programa con las capacidades de tiempo real de RTAI, se programó un código en C para correrlo en user-space.

- Creación de la Tarea de tiempo real: para permitir que una tarea se ejecute utilizando las capacidades de RTAI únicamente es necesario crear una tarea con la función dedicada *rt_task_init(name, priority, stack_size, max_msg_size)*, en esta función en el espacio *priority* el valor *0* le asigna la mas alta prioridad a la tarea creada.
- Inicialización del timer: el timer es el principal paso para permitir una restricción de tiempo determinística en la tarea creada en RTAI. El temporizador puede ser iniciado o detenido con las siguientes APIs: *start_rt_timer(period)*, *stop_rt_timer()*.
- Modo Oneshot vs Modo Periodico: el modo Oneshot consiste en un temporización variable basada en la frecuencia del CPU. El Modo Periódico consiste en una temporización fija de las tareas, como un múltiplo del periodo establecido con la función *start_rt_timer(period)*.
- Ejecutar una tarea: en este punto el temporizador está corriendo con el periodo apropiado y de acuerdo al modo escogido(Oneshot o Periódico), para ejecutar la tarea en tiempo real se usa la función *rt_task_make_periodic(task, start_time, period)*. La tarea puede ser ejecutada en tiempo real estricto o flexible con las funciones *rt_make_hard_real_time()* y *rt_make_soft_real_time()*.

- RTAI ofrece mas APIs para diferentes aplicaciones.

Las anteriores funciones mas algunas otras secundarias deben ser escritas, utilizando las correspondientes librerías. [15]

Un ejemplo del código utilizado para correr una tarea de adquisición de datos y control control, en tiempo real se muestra en los anexos.

4.2 Comprobación del SAC por medio de un Sistema Mecánico

Para la comprobación del Sistema de Administración y Control se creó un sistema mecanico simple (planta), el cual consta de dos Motores DC y una Caja de Engranajes Reductora de Velocidad (Gearbox) para cada Motor. La planta se muestra en la figura 4, en esta se indican, los Gearboxes (una para cada motor) y los Motores CD, además se muestra la Variable de Estado de Posición Angular (θ) de uno de los ejes del Gearbox, que es movido por uno de los motores CD, esta es una de las variables que se controlan, como se explicará mas adelante.



Figura 4. Sistema Mecánico a controlar.

El Gearbox fue ensamblado de manera que la Relación de Reducción fuera de 203:1, es decir por cada 203 vueltas del eje del Motor CD, el eje del Gearbox gira solamente una vez.

Un diagrama de bloques del Sistema Robótico a controlar se muestra en la figura 5

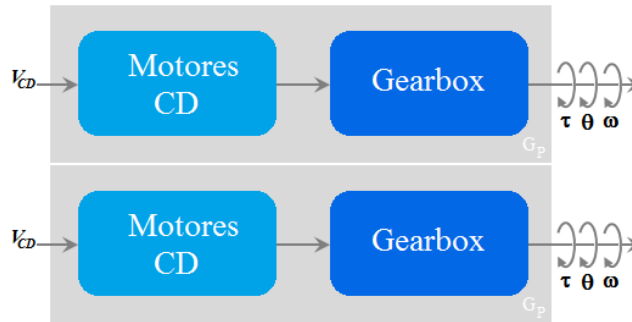


Figura 5. Diagrama de bloques del sistema dinámico a controlar

En la figura 5 se muestran las Variables Estado del Sistema Dinámico, estas son: *Torque* (τ), *Velocidad Angular* (ω), y *Posición Angular* (θ).

4.3 Esquema General del Control del Sistema Mecánico

4.3.1 Sistema de Control

Un *Sistema de Control* consta de dos partes esenciales, el *Controlador*, que es diseñado para cumplir las especificaciones requeridas, y la *Planta*, que es el Sistema Mecánico que se quiere controlar; la figura 6 muestra el diagrama de bloques del Sistema de Control.

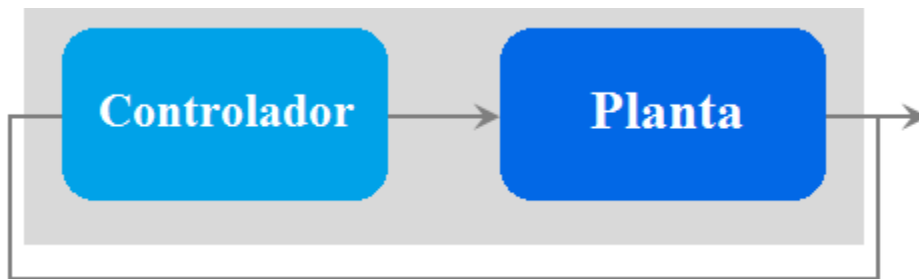


Figura 6. Sistema de Control.

4.3.2 Planta (G_P)

La *Planta* que se quiere controlar es la indicada en la sección 4.2.

4.3.3 Controlador (G_C)

El *Controlador* (G_C) en este caso es el SAC explicado en la sección 4.1. Además del SAC fue necesario el diseño de una etapa de potencia, para que el Sistema Mecánico pueda utilizar las señales generadas por la SAC, la señal de salida de la *Etapa de Potencia* es V_{CD} , esta es la señal que alimenta a los Motores CD (una por Motor CD). En la figura 7 se muestra el diagrama de bloques del *Controlador*.

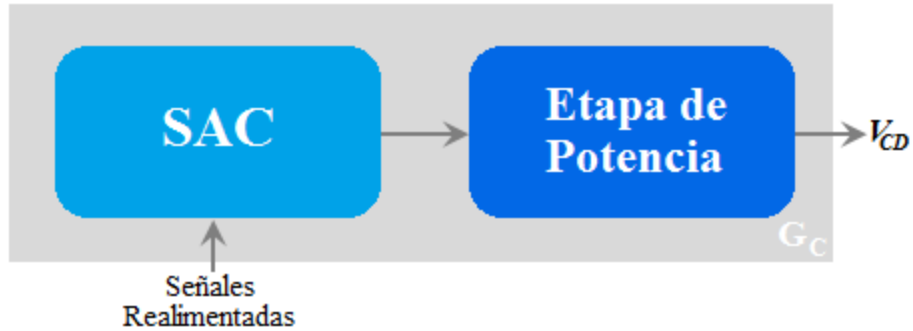


Figura 7. Diagrama de Bloques del Controlador.

4.3.4 Etapa de Potencia

La *Etapa de Potencia* está compuesta por un *Convertidor CD/CD Buck Reductor*, controlado por *PWM* (Modulador de Ancho de Pulso).

Esta etapa utiliza la señal analógica entregada por el SAC (V_{GC}), para controlar la conmutación del Buck, con la conmutación del Buck permite variar la tensión del Motor CD (V_{CD}), para alcanzar distintas velocidades. En la figura 8 se muestra el diagrama de bloques de la Etapa de Potencia de uno de los Motores de la planta.

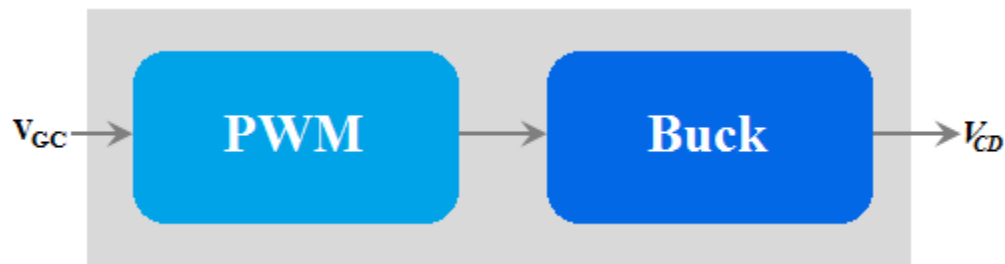


Figura 8. Etapa de Potencia.

4.3.5 PWM

Genera una señal cuadrada con un Ciclo de Trabajo⁸ variable, controlado por una señal CD de referencia, que en el caso del Controlador es V_{GC} , y una señal triangular, el esquema del PWM utilizado se presenta en la figura 9. [16]

⁸ El ciclo de trabajo D se define como la relación entre la duración del pulso(τ) y el periodo de una onda rectangular(T), $D = \frac{\tau}{T}$.

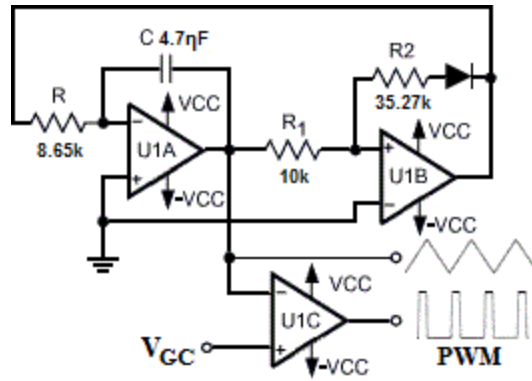


Figura 9. Circuito generador de PWM.

El operacional U1A actúa como un integrador, el cual proporciona a su salida una señal de forma de una ecuación lineal, el U1A es responsable de generar las rectas que conforman la onda triangular, el operacional U1B actúa como un comparador, el cual se encarga de cambiar el la polaridad del voltaje CD que es integrado por U1A, por ende, este operacional es responsable de los cambios de pendiente positiva a pendiente positiva en la onda triangular.

Para dimensionar el circuito de la figura 9, se siguió el siguiente proceso:

Primero se calculó el factor p , con $V_{sat}=10V$:

$$p = -\frac{-V_{sat} + 0.6V}{V_p} = 3.1333$$

Se quiere una frecuencia de conmutación de $F_s = 40kHz$, el valor de R para lograr esta frecuencia se calcula con la siguiente ecuación, se escogió $C=4.7nF$:

$$R \cong \frac{p}{2fC} \cong 8.3333k\Omega$$

Y el valor de R2 se calculó con la siguiente relación:

$$R2 = pR = 31.33k\Omega$$

Realizando ajustes a R y pR, sustituyendo las resistencias por potenciómetros, se obtuvieron los siguientes valores:

$$R \cong 8.65k\Omega \quad R2 = 35.27k\Omega$$

El chip utilizado fue un TL084, este tiene un slew rate de $13 V/\mu s$, lo que le permite trabajar a la frecuencia requerida por la onda triangular y por el PWM.

Además cuenta con cuatro OpAmps, lo que permitió implementar el generador de onda triangular con dos de los Op Amps, y con los dos Op Amps restantes se implementó un comparador para cada Motor CD.

El circuito de la figura 9 fue ajustado para proporcionar una señal triangular de 3Vpp CD, para que coincida el máximo valor de tensión aceptable por los Motores CD con el ciclo de trabajo de 100%, con una frecuencia de aproximadamente 40kHz, y una señal cuadrada (PWM) de 20Vpp, que corresponden a las tensiones de saturación del Operacional que fue alimentando con $\pm 10V$ a una frecuencia de 40kHz y de ciclo de trabajo variable.

4.3.6 Convertidor Buck

El convertidor Buck, mostrado en la figura 10, transforma una tensión de entrada V_{in} , en una tensión menor de salida V_{out} , con este fin, el convertidor es controlado con un PWM. Puede ser demostrado que en régimen permanente la tensión de salida (V_{out}) depende linealmente del Ciclo de Trabajo (D), es decir:

$$V_{out} = D * V_{in}, \quad 0 \leq D \leq 1$$

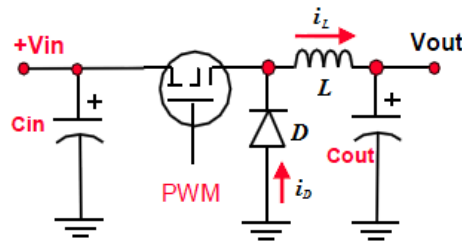


Figura 10. Convertidor Buck.

El convertidor Buck tiene dos modos de operación. En el primer modo el MOSFET está “encendido”, en este modo, la corriente en la bobina (L) aumenta, cuando el MOSFET es apagado, la corriente en la bobina se reduce. La figura 11 muestra diagramas de tensión y corriente típicos de un Convertidor Buck, cuando es usado en el llamado modo continuo, es decir cuando $i_L > 0$ siempre.

Debido a este modo de operación cambiante, la corriente en la bobina experimenta un rizado Δi_L . Pero a diferencia del caso cuando un PWM es usado directamente como alimentación del Motor, para generar una corriente promedio deseada, el rizado de corriente en un Convertidor Buck puede ser asignado a un valor pequeño, con una elección apropiada de la inductancia de L . Para un Condensador C_{out} correctamente dimensionado, la tensión de salida V_{out} , puede asumirse constante.

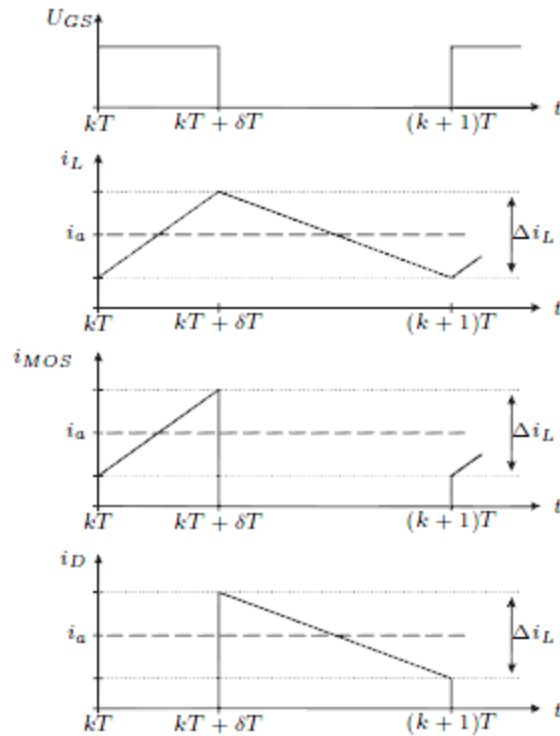


Figura 11. Tensiones y Corrientes en el Buck para el Modo Continuo

La frecuencia de conmutación escogida previamente en el PWM fue de $F_s = 40\text{KHz}$, un $V_{in} = 3\text{V}$ fue escogido, debido a que es la tensión máxima del Motor CD.

4.3.7 Elección de los componentes del Convertidor Buck

Como fue mencionado anteriormente, el rizado en la corriente Δi_L depende de la inductancia de la bobina, la relación es dada por:

$$\Delta i_L = \frac{1}{L} (V_{in} - V_{out}) \frac{D}{F_s}$$

Con el propósito de asegurar que el Convertidor siempre opere en Modo Continuo, la corriente de salida promedio mínima se toma como la corriente del Motor CD cuando gira a velocidad constante. Para el Motor CD en este caso, se midió la corriente a Tensión Nominal ($V = 3\text{V}$) y la Corriente del Motor fue de $i_M = 215\text{mA}$, esta misma corriente i_M ha sido escogida como Δi_L , es decir $\Delta i_L = 215\text{mA}$. La inductancia necesaria de acuerdo a la ecuación anterior también depende de V_{out} , la cual varía. Entonces para asegurar $\Delta i_L < 215\text{mA}$, para todas las tensiones posibles de salida, se escogió $V_{out} = \frac{V_{in}}{2}$ lo cual maximiza el lado derecho de la ecuación anterior. Por lo tanto la inductancia necesaria resulta ser:

$$L = \frac{V_{in}}{4F_s} \frac{1}{\Delta i_L} = 87 \mu H$$

Un valor comercial de aproximadamente $116 \mu H$ se tenía disponible, y para este valor el rizado de la corriente de salida resulta ser:

$$\Delta i_L = \frac{V_{in}}{4F_s} \frac{1}{L} = 161 mA$$

El cual es un valor aun aceptable para que el Convertidor Buck opere en modo continuo.

Basándose en las curvas de la figura 11(Buck), el MOSFET utilizado como interruptor debe soportar una corriente máxima de Drain(i_D), y una tensión de Gate-Source(V_{GS}) de:

$$i_{MOS} = i_a + \frac{\Delta i_L}{2} = 215 mA + \frac{116 mA}{2} = 273 mA$$

$$V_{GS} = \pm 10V$$

Por lo tanto se escogió el NTE2371, que es un MOSFET de canal P, con las siguientes características:

$$i_D = 19 A_{T_c = +25^\circ C} \quad V_{GS} = \pm 20V$$

Según la curva de corriente de la figura # (las del Buck), el diodo debe soportar una corriente pico de:

$$i_D = i_a + \frac{\Delta i_L}{2} = 273 mA$$

Además debe ser de conmutación rápida, por lo tanto se eligió el diodo Shottky NTE585, que puede soportar una corriente de hasta 1A.

La bobina y el condensador a la salida del Convertidor forman un filtro pasa bajas que puede filtrar perturbaciones de alta frecuencia causadas por la conmutación en el MOSFET. La frecuencia de corte del filtro es dada por:

$$f_{co} = \frac{1}{2\pi\sqrt{LC}}$$

Para asegurar poco rizado en la tensión de salida del Buck, se debe hacer $f_{co} \ll F_s$, si se escoge $f_{co} = 1 kHz$ se obtiene $C = 218.37 \mu F$, por lo tanto se escogió un valor comercial cercano de $C = 220 \mu F$. [17]

Se hicieron dos Buck Converter, uno para cada Motor CD.

4.3.8 Sensor de posición

La señal retroalimentada es la *Posición Angular* (θ) del eje del Gearbox, como se muestra en la figura 4(planta), esta se midió con el **Encoder Magnético Giratorio Absoluto MAE3** de la empresa USDigital, este encoder proporciona información de la posición del eje sobre 360° de rotación, con una salida analógica entre 0V para la posición 0° y 5V para la posición 360°, la señal analógica de salida del MAE3 de acuerdo a la posición del eje, se muestra en la figura 12.

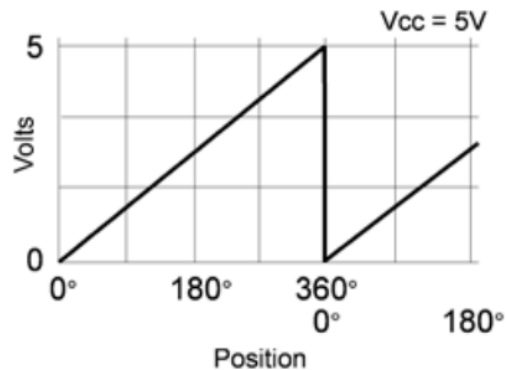


Figura 12. Señal analógica de salida del Encoder MAE3 vs la posición angular.

El encoder funciona por un imán que se une al eje del Gearbox y a un sensor magnético que se encarga de convertir el campo magnético producido por el imán, en una tensión proporcional a la *Posición Angular*. En la figura 13 se muestra el Encoder Magnético Giratorio Absoluto MAE3 y sus partes.



Figura 13. Encoder Magnético Giratorio Absoluto.

Para poder ubicar el Encoder en una posición apropiada se debió construir una base para los Motores CD, el Gearbox y los encoders, esta se muestra en la figura 14.



Figura 14. Base para los sensores de la *Planta*.

4.4 Esquema general del Sistema de Control

Un diagrama de bloques del Sistema de Control completo se muestra en la figura 15.

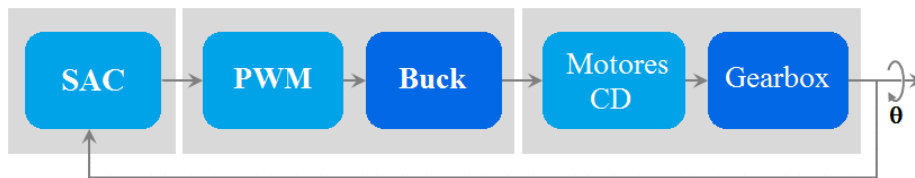


Figura 15 Diagrama de Bloques del Sistema de Control completo.

4.5 Modelado Matemático

En la figura 16 se presenta el *Modelo Eléctrico y Mecánico* del motor CD, junto con el Gearbox.

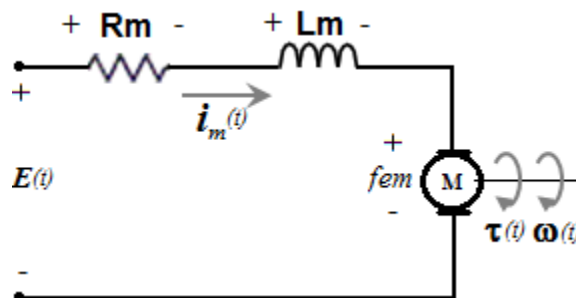


Figura 16. Modelo del Motor CD y el Gearbox.

A partir del modelo de la figura 16 se obtuvieron las siguientes *Ecuaciones de Estado*.

$$\frac{di_m(t)}{dt} = -\frac{R_m}{L_m}i_m(t) - \frac{k_e}{L_m}\omega(t) + \frac{1}{L_m}V(t)$$

$$\frac{d\omega(t)}{dt} = \frac{k_T}{J}i_m(t) - \frac{B}{J}\omega(t)$$

$$\frac{d\theta(t)}{dt} = \omega(t)$$

De las ecuaciones de estado se obtienen las siguientes matrices

$$A = \begin{bmatrix} -\frac{R_m}{L_m} & -\frac{k_e}{L_m} & 0 \\ \frac{k_T}{J} & -\frac{B}{J} & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} \frac{1}{L_m} \\ 0 \\ 0 \end{bmatrix} \quad C = [0 \quad 0 \quad 1]$$

A partir de las Matrices A, B y C se pueden obtener la función de transferencia de lazo abierto, entre la *Posición Angular* del eje del sistema dinámico ($\Theta(s)$), y la *Tensión de Entrada* ($V(s)$), la misma se presenta a continuación. [18]

$$\frac{\Theta(s)}{V(s)} = \frac{k_T}{s^3 + \frac{J R_m + B L_m}{J L_m} s^2 + \frac{k_T K_e + B R_m}{J L_m} s}$$

Además con las matrices A y B, y utilizando el software Maple⁹, se obtiene la matriz de controlabilidad (\mathbf{R}), esta se presenta a continuación:

$$\mathbf{R} = \begin{pmatrix} \frac{1}{L_m} & -\frac{R_m}{L_m^2} & \frac{R_m^2}{L_m^3} - \frac{k_e k_t}{L_m^2 J} \\ 0 & \frac{k_t}{J L_m} & -\frac{k_t R_m}{J L_m^2} - \frac{B k_t}{J^2 L_m} \\ 0 & 0 & \frac{k_t}{J L_m} \end{pmatrix}$$

El rango de la matriz \mathbf{R} calculado con el software Maple da como resultado 3, lo cual corresponde al número de columnas y filas de la matriz \mathbf{R} , por lo tanto el sistema es controlable.

⁹ Maple es un programa para computadora que puede realizar cálculos simbólicos y algebraicos.

4.6 Caracterización De la Planta

Para el diseño del *Controlador* de la planta es necesario caracterizar la misma, se deben obtener los valores físicos del *Coefficiente de Fricción Viscosa* (B), el *Momento de Inercia* (J); ambos incluyen la fricción del Motor CD y del Gearbox, como también las *constantes de torque* (k_T) y *de tensión* (k_e) del Motor CD.

Para obtener la *Constante de Torque* (k_T) y la *Constante de Tensión* (k_e) del Motor CD se utilizó la hoja de datos proporcionada por el fabricante que se presenta en la figura 17.

La curva de interés es la que se grafica para Torque (TORQUE) vs Corriente (I), y la ecuación del par inducido en un Motor CD de imanes permanentes, como es el caso de los motores con los que se trabajó es:

$$\tau = \frac{2}{\pi} \phi i$$

En la que el factor $\frac{2}{\pi} \phi$ es constante inherente al motor, y en este caso es llamada la Constante de Torque, y en la figura 17 corresponde a la pendiente de la recta.

Para determinar el valor numérico de k_T , se escogieron dos puntos en la recta, marcados con rojo en la figura 17, y se aplicó la ecuación de la pendiente para una ecuación lineal, tomando el torque como la variable dependiente y la corriente como la independiente. La misma se muestra a continuación:

$$k_T = \frac{1 \times 10^{-3} \text{ N}\cdot\text{m} - 250 \times 10^{-6} \text{ N}\cdot\text{m}}{700 \text{ mA} - 300 \text{ mA}} = \frac{750 \times 10^{-6} \text{ N}\cdot\text{m}}{400 \text{ mA}} = 1.875 \times 10^{-3} \frac{\text{N}\cdot\text{m}}{\text{A}}$$

Si las constantes del motor se expresan en unidades de SI, su valor numérico es el mismo, por lo tanto:

$$k_T = 1.875 \times 10^{-3} \frac{\text{N}\cdot\text{m}}{\text{A}} \qquad k_e = 1.875 \times 10^{-3} \text{ V}\cdot\text{s}$$

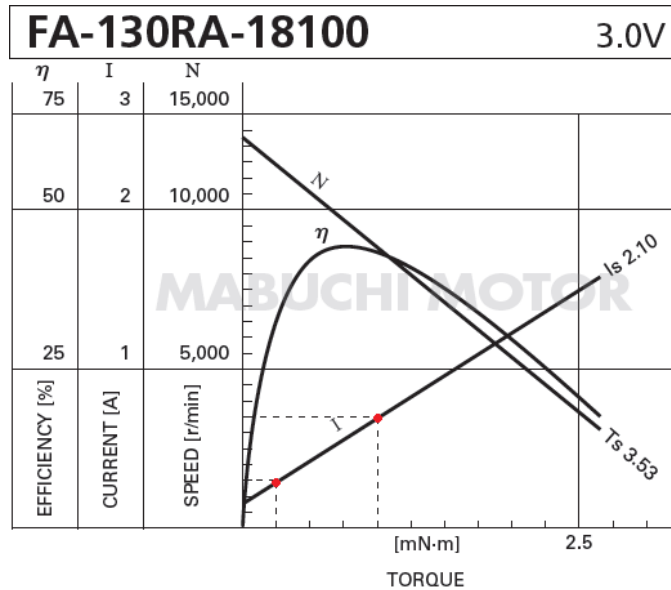


Figura 17. Curvas del Motor CD.

Para obtener el *Coefficiente de Friccion Viscosa (B)*, se necesita medir la *Velocidad Angular (ω)* del eje del Gearbox, la medición se hizo utilizando un *Encoder Incremental* de la marca USDigital, específicamente el modelo E2 con un disco de 900 ciclos por revolución, es decir, proporciona 900 pulsos cuadrados por cada vuelta del eje del Gearbox, el encoder E2 se presenta en la figura 18:

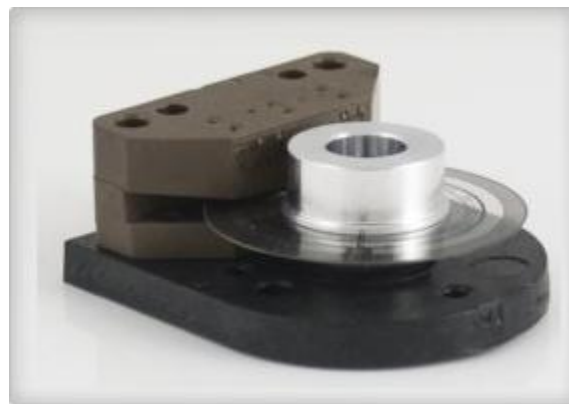


Figura 18. Encoder Incremental E2.

La base que se muestra en la figura 14, también fue utilizada para colocar el Encoder Incremental E2.

Para medir la velocidad del eje del Gearbox se alimentó el Motor CD con 3V, y se visualizó en un osciloscopio el tren de pulsos entregado por el encoder, en la figura 19 se muestra el tren de pulsos.

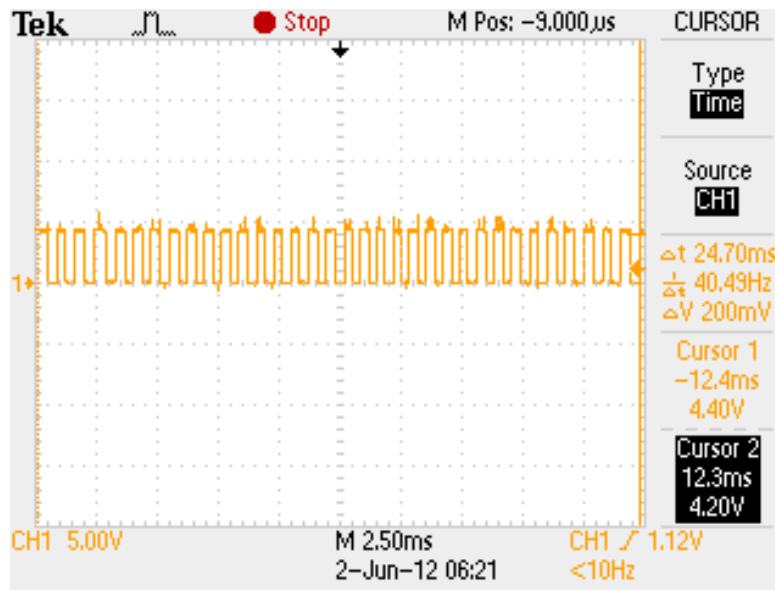


Figura 19. Tren de pulsos entregado por el Encoder Incremental.

La Velocidad Angular (ω), y en este caso la nominal, del eje del Gearbox se dedujo a partir de la cantidad de pulsos cuadrados, en un espacio de tiempo, en la figura 19 se cuentan 34 pulsos(p) en 24.7ms, para obtener la velocidad en radianes por segundo (rad/s) se hicieron los siguientes cálculos:

$$\frac{900p}{34p} \rightarrow \frac{1 \text{ rev}}{x \text{ rev}}$$

$$x = 37.77 \times 10^{-3} \text{ rev}$$

$$\omega = \frac{37.77 \times 10^{-3} \text{ rev}}{24.7 \text{ ms}} = 1.57 \text{ rps} = 9.6 \frac{\text{rad}}{\text{s}}$$

Además de la velocidad se midió la corriente del Motor alimentado con 3V, el resultado fue:

$$i_m = 215 \text{ mA}$$

Con los datos obtenidos anteriormente se puede calcular B por medio de la siguiente relación:

$$B = \frac{k_T \cdot i}{\omega} = \frac{(250 \text{ mA}) \left(1.875 \times 10^{-3} \frac{\text{N} \cdot \text{m}}{\text{A}} \right)}{9.6 \frac{\text{rad}}{\text{s}}} = 48.8281 \times 10^{-6} \text{ N} \cdot \text{m} \cdot \text{s}$$

Para medir el *Momento de Inercia* (J) es necesario medir la *Constante Mecánica de Tiempo* (τ_m), para su medición se utilizó nuevamente el *Encoder Incremental E2* y un convertidor Frecuencia-Voltaje, de forma que permita ver la curva de velocidad del motor, interpretado por medio de una tensión.

El convertidor Frecuencia-Voltaje utilizado fue el LM2907, el mismo se conectó como se muestra en al figura 20:

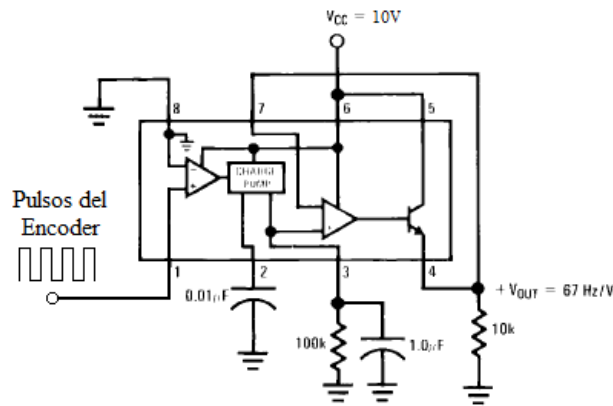


Figura 20. Convertidor Frecuencia-Voltaje.

Para ver la curva de velocidad del motor se le aplicó un escalón de 3V(tensión nominal) a uno de los Motores CD y se observó en el osciloscopio el cambio de la velocidad(tensión) en la salida del convertidor Frecuencia-Voltaje, la curva de velocidad del motor se muestra en la figura 21:



Figura 21. Curva de velocidad del Motor CD y Gearbox.

En la figura 21 se puede ver que el tiempo que tarda el motor en llegar al 63.2% (equivalente a 5.36V en la fig. 21) de su velocidad nominal es de 196ms, este valor es la Constante Mecánica de Tiempo.

Para calcular el momento de Inercia del Motor y el Gearbox se utilizó la siguiente expresión:

$$J = \tau_m B = 9.5703 \times 10^{-6} \text{ N}\cdot\text{m}\cdot\text{s}^2$$

Los resultados obtenidos de la caracterización fueron [\[19\]](#)

$$k_T = 1.875 \times 10^{-3} \frac{\text{N}\cdot\text{m}}{\text{A}} \quad k_e = 1.875 \times 10^{-3} \text{ V}\cdot\text{s}$$

$$B = 48.8281 \times 10^{-6} \text{ N}\cdot\text{m}\cdot\text{s} \quad J = 9.5703 \times 10^{-6} \text{ N}\cdot\text{m}\cdot\text{s}^2$$

La función de transferencia de la planta real es:

$$\frac{\Theta(s)}{V(s)} = \frac{541510.8436}{s^3 + 2492.6642 s^2 + 13706.99 s}$$

El ancho de banda de esta función se determinó mediante la magnitud del diagrama de Bode, para el valor en que la Magnitud es de -3dB, y tiene un valor de $BW = 2.72\text{Hz}$.

Según el teorema de Nyquist, se necesita un periodo de muestreo de mínimo

$$T_s \geq 2 \cdot BW \geq 735.29 \times 10^{-3}$$

Esta dato es importante para demostrar que RTAI puede controlar a este sistema, como se explica en la sección de [análisis de resultados](#).

4.7 Diseño del Controlador de Posición

En esta sección se muestra el proceso de *Modelado Matemático* de uno de los motores CD junto con el Gearbox, y el diseño del *Controlador de Posición Angular*, referido a partir de ahora como C_P

En la figura 22 se muestra el diagrama de bloques para el *Controlador de Posición*.

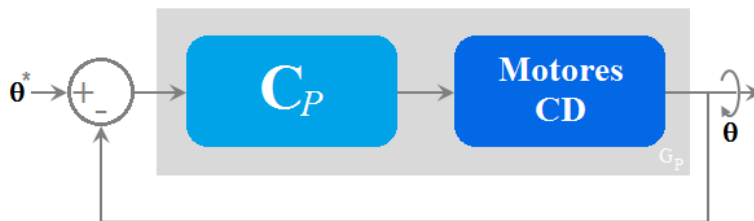


Figura 22. Diagrama de Bloques del Controlador de Posición.

Los requisitos del Controlador a diseñar son:

1. No sobreimpulso.
2. Error en Régimen Permanente igual a cero.
3. Tiempo de asentamiento menor a 3s.

Utilizando un *Control Proporcional*, $C_P = K$, se obtiene la siguiente ecuación a lazo cerrado, la K mayúscula es la constante de Control Proporcional.

$$G_{CL} = \frac{k_T K}{s^3 + \frac{J R_m + B L_M}{J L_m} s^2 + \frac{k_T K_e + B R_m}{J L_m} s + K k_T}$$

Y utilizando los valores obtenidos en la caracterización se obtiene:

$$G_{CL} = \frac{541.59 \times 10^{-3} K}{s^3 + 2.5 \times 10^3 s^2 + 13.7 \times 10^3 s + K 541.594 \times 10^3}$$

4.7.1 Respuesta en Régimen Permanente

El cálculo del *Error en Régimen Permanente* (e_{ss}) de la respuesta de la *planta* con un *Control Proporcional*, y ante una *entrada escalón unitario*, se obtuvo mediante la ecuación de Transferencia de *Trayectoria Directa*:

$$e_{ss} = \frac{1}{1 + G_{td}} = \frac{1}{1 + \frac{K k_t}{0}} = \frac{1}{1 + \infty} = \frac{1}{\infty} = 0$$

4.7.2 Estabilidad del Sistema de Control de Posición

Para estudiar la estabilidad se utilizó el método de la tabla de Hurwitz, la tabla 1 presenta los resultados de este método:

s^3	1	11.47×10^3
s^2	2.49×10^3	$419.3 \times 10^3 K$
s^1	$11.47 \times 10^3 - 168.27 K$	0
s^0	$419.3 \times 10^3 K$	0

Tabla 1. Tabla de Hurwitz.

El criterio de Hurwitz establece que todos los números en la primer columna (columna sombreada) deben ser positivos, para que el sistema sea estable, por este motivo se hace necesario calcular los valores de K para los que se cumple el criterio de Hurwitz, es decir los valores de K para los que se cumple:

$$11.47 \times 10^3 - 168.27 K > 0 \qquad 419.3 \times 10^3 K > 0$$

De las ecuaciones anteriores se obtiene:

$$0 < K < 68.1643$$

Para que el sistema sea estable. [20]

4.7.3 Respuesta transitoria

Con la ayuda de la herramienta SISO Tool de Matlab, se obtuvo el lugar de las raíces del sistema en la Lazo Cerrado con control K , en la figura 23 se presenta el resultado.

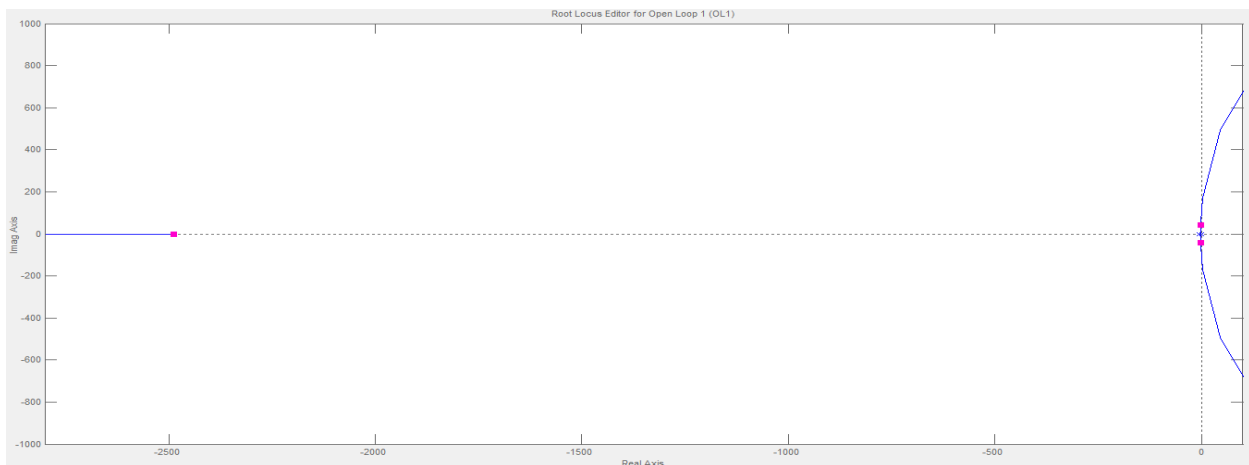


Figura 23. Lugar de las raíces para el sistema en lazo cerrado, con un control K .

Como se puede ver en la figura 23 el sistema tiene un polo real, y un par complejo conjugado que pasa muy cerca del origen, por lo que la respuesta transitoria se ve principalmente afectada por ellos.

Se analizó la posición que debe tener el par complejo conjugado para cumplir con el requisito de no sobre impulso, el lugar de las raíces y la respuesta al escalón del sistema obtenida con Matlab se presentan en la figura 24.

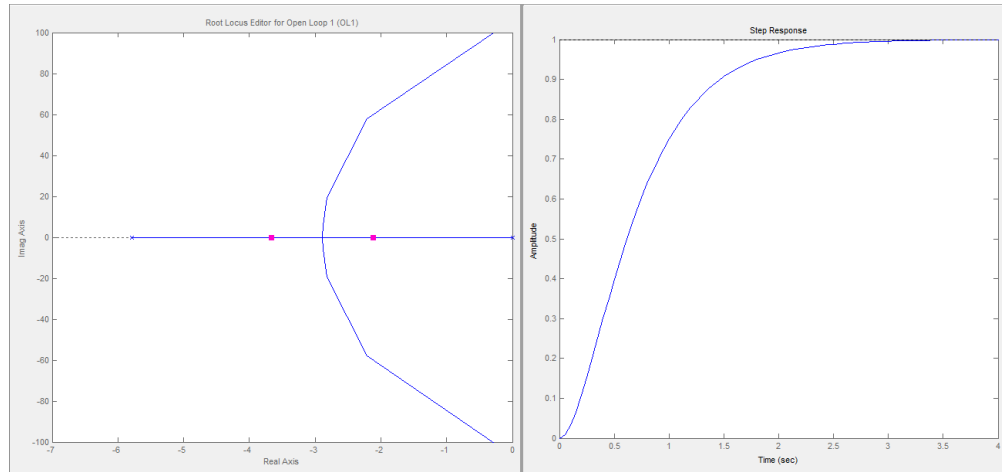


Figura 24. Lugar de las raíces y respuesta al escalón para cumplir el requisito de no sobre impulso y tiempo de asentamiento menor o igual a 3s, con $K = 0.03$.

Como se puede ver en la figura 24, con $K = 0.03$ el par complejo conjugado se ubica sobre el eje real, lo que corresponde a un sistema sobreamortiguado, el cual no tiene sobre impulso, por lo que ese valor de K fue el escogido para el controlador.

4.7.4 Implementación del controlador en el SAC

Como se mencionó en el marco teórico, la ventaja de tener un sistema computacional para implementar controladores es la rapidez y facilidad con que el controlador puede ser implementado, en el caso del Sistema Mecánico anteriormente expuesto se debe implementar una lectura de la información proporcionada por el Encoder Magnético, una función que se encargue de procesar la información leída y posteriormente una función que se encargue de escribir en una salida analógica la información procesada.

Las funciones anteriormente descritas se implementaron con los siguientes comandos de software:

- **Funciones de lectura de los encoders:**

```
comedi_data_read(it,subdev,3,range,aref, &M_pos_1);  
comedi_data_read(it,subdev,4,range,aref, &M_pos_2);
```

Ambas funciones toman un dato analógico entregado por el encoder, lo digitalizan y lo almacenan en las variables M_pos_1 y M_pos_2 .

- **Procesamiento de la información leída**

Punto al que se quiere llegar el eje de los motores:

```
m_pos_set_1 = 47000; //Punto al que debe llegar el eje del Motor 1  
m_pos_set_2 = 47000; //Punto al que debe llegar el eje del Motor 2
```

Calculo del error de posición:

```
e_m_pos_1 = ( m_pos_set_1 - (M_pos_1)); //Calculo del Error de posición  
e_m_pos_2 = ( m_pos_set_2 - (M_pos_2)); // Calculo del Error de posicion
```

Multiplicacion por la constante proporcional **K**:

```
V_GC1 = K*e_m_pos_1 + 32750 ;  
V_GC2 = K*e_m_pos_2 + 32750 ;
```

Escritura de las señales de control V_{GC} en las salidas analógicas:

```
comedi_data_write(it,1,0,range,aref, V_ref_2); //escribe V_GC2  
comedi_data_write(it,1,1,range,aref, V_ref_1); //escribe V_GC1
```

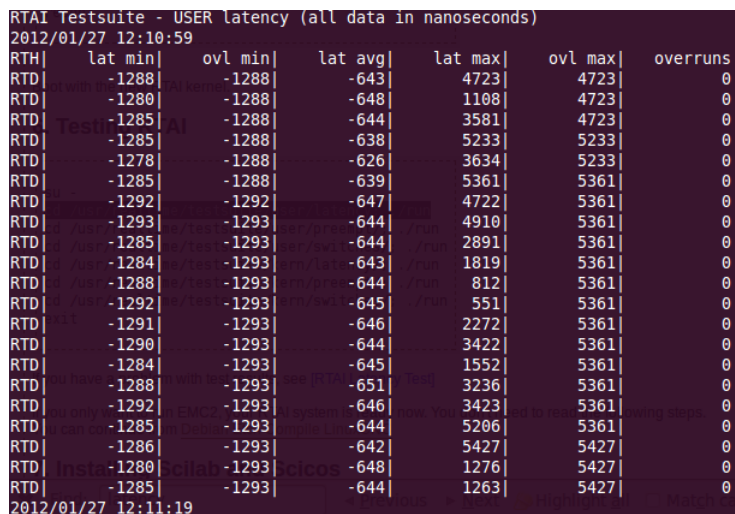
Es claro que la implementación de un controlador en una plataforma de control basada en una computadora personal es mucho mas rápida y eficiente que la implementación de un controlador analógico, además de que el ajuste de este tipo de controladores es mas rápido y sencillo ya que los parámetros como **K** pueden ser cambiados de forma fácil en el programa del controlador.

5 Resultados

La instalación del parche de software RTAI con el kernel de Linux es un proceso lento y tedioso, principalmente para una persona con pocos conocimientos de Linux y software, según lo visto en foros, la instalación de RTAI se vuelve tediosa inclusive hasta para personas con experiencia en Linux. A pesar de existir diversos tutoriales que explican como lograr la correcta instalación, estos no necesariamente son de utilidad, ya que fueron hechos para instalar RTAI en diferentes distribuciones de Linux, u orientados a diferentes arquitecturas de procesador, o para versiones de las distribuciones, antiguas de Linux, incluso el mismo tutorial proporcionado en el sitio web oficial de RTAI es bastante antiguo, y algunos de los pasos que ahí se indican no son claros.

El primer intento de instalar RTAI se hizo con la distribución Debian, la cual es mas estable que Ubuntu, sin embargo tras varias semanas de tratar de compilar el kernel junto con RTAI sin éxito, se optó por probar con la distribución Ubuntu, tras semanas de ensayo y error, tratando de instalar exitosamente el parche, se logró mediante un tutorial encontrado en la página www.rtaixml.net.

El proceso completo de instalar el kernel de Linux con el parche de RTAI tomó aproximadamente mes y medio. Al final para comprobar que RTAI ha sido instalado correctamente se procedió a hacer la prueba de latencia, esta es ofrecida por el mismo paquete de RTAI, los resultados de la prueba de latencia se muestran en el screenshot de la figura 25.



```
RTAI Testsuite - USER latency (all data in nanoseconds)
2012/01/27 12:10:59
RTH|   lat min|   ovl min|   lat avg|   lat max|   ovl max|  overruns
RTD|   -1288|   -1288|   -643|   4723|   4723|  0
RTD|   -1280|   -1288|   -648|   1108|   4723|  0
RTD|   -1285|   -1288|   -644|   3581|   4723|  0
RTD|   -1285|   -1288|   -638|   5233|   5233|  0
RTD|   -1278|   -1288|   -626|   3634|   5233|  0
RTD|   -1285|   -1288|   -639|   5361|   5361|  0
RTD|   -1292|   -1292|   -647|   4722|   5361|  0
RTD|   -1293|   -1293|   -644|   4910|   5361|  0
RTD|   -1285|   -1293|   -644|   2891|   5361|  0
RTD|   -1284|   -1293|   -643|   1819|   5361|  0
RTD|   -1288|   -1293|   -644|   812|   5361|  0
RTD|   -1292|   -1293|   -645|   551|   5361|  0
RTD|   -1291|   -1293|   -646|   2272|   5361|  0
RTD|   -1290|   -1293|   -644|   3422|   5361|  0
RTD|   -1284|   -1293|   -645|   1552|   5361|  0
RTD|   -1288|   -1293|   -651|   3236|   5361|  0
RTD|   -1292|   -1293|   -646|   3423|   5361|  0
RTD|   -1285|   -1293|   -644|   5206|   5361|  0
RTD|   -1286|   -1293|   -642|   5427|   5427|  0
RTD|   -1280|   -1293|   -648|   1276|   5427|  0
RTD|   -1285|   -1293|   -644|   1263|   5427|  0
2012/01/27 12:11:19
```

Figura 25. Resultado de la prueba de latencia de RTAI.

En la figura 25 se puede observar en la columna de mas a la derecha, que el número de overruns es 0, lo cual es muestra de que RTAI está siendo ejecutado de manera correcta, además la latencia máxima muestra resultados del orden de microsegundos, es decir que le toma a RTAI

microsegundos completar tareas asignadas con alta prioridad; esta latencia es suficiente para controlar sistemas mecánicos.

Una vez superada la etapa de instalación de RTAI se procedió a programar en C la rutina que se encarga de controlar la DAQ para obtener datos de sus entradas analógicas y escribir datos en las salidas analógicas, posteriormente se programó la misma rutina, pero utilizando las capacidades de RTAI, el programa descrito anteriormente se corrió en User Space.

Lo anterior preparó la plataforma sobre la cual se programa el control de un sistema mecánico, en el caso particular de este proyecto se programó el controlador proporcional para la posición de la planta descrita en las secciones anteriores.

La onda triangular para el PWM de la Etapa de Potencia se muestra en la figura 26, este tiene una amplitud de 3Vpp y una frecuencia de aproximadamente 40KHz, suficientemente para sistemas mecánicos.

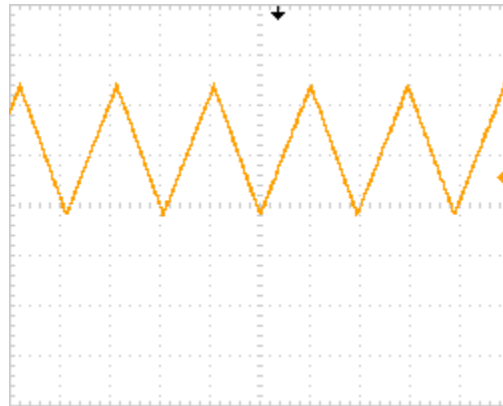


Figura 26. Onda triangular del PWM.

El circuito generador de la onda triangular inicialmente fue dimensionado para que tuviera una amplitud pico a pico de 5V, posteriormente fue redimensionado para tener una amplitud de 3Vpp, lo cual se tradujo en el sistema real con una respuesta más rápida ante un escalón.

La onda cuadrada del PWM se muestra en la figura 27, diferentes valores de V_{GC} fueron usados para obtener los diferentes ciclos de trabajo.

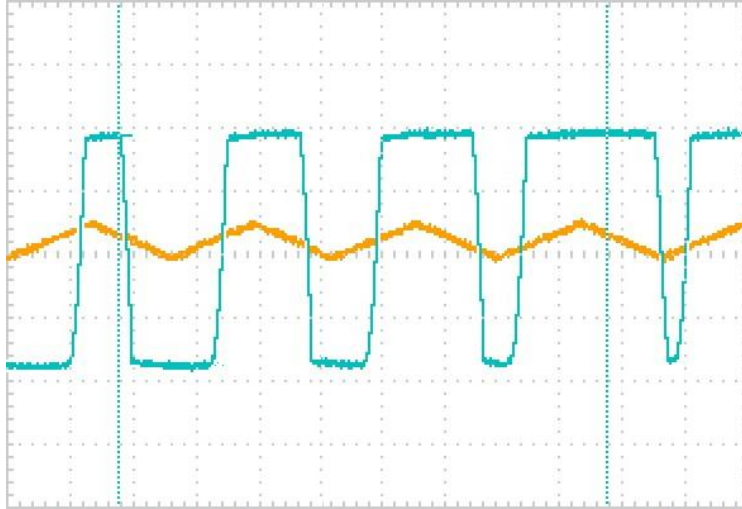


Figura 27. PWM con diferentes ciclos de trabajo.

La respuesta de uno de los Convertidores DC/DC Buck ante una entrada escalón, la cual se implementa por medio de un cambio repentino en el ciclo de trabajo del PWM, de la figura anterior se muestra en la figura 28.

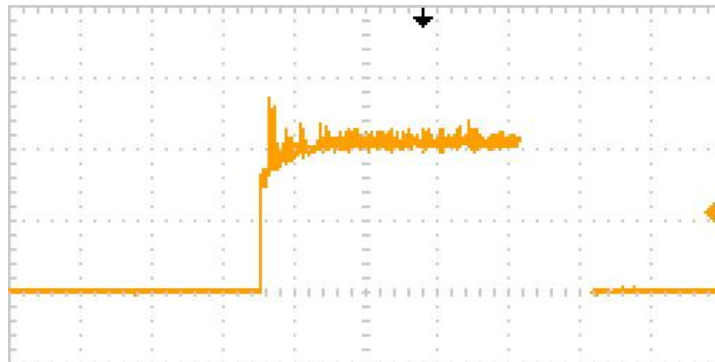


Figura 28. Respuesta del Convertidor Buck ante un escalón.

La curva de la figura 28 fue obtenida con el Buck operando con el Motor DC como carga.

En la figura 29, se muestra un la respuesta al escalón del sistema real, la curva es de la posición angular, representada por la tensión entregada por el Encoder Magnético Absoluto MAE3, variando en el tiempo.

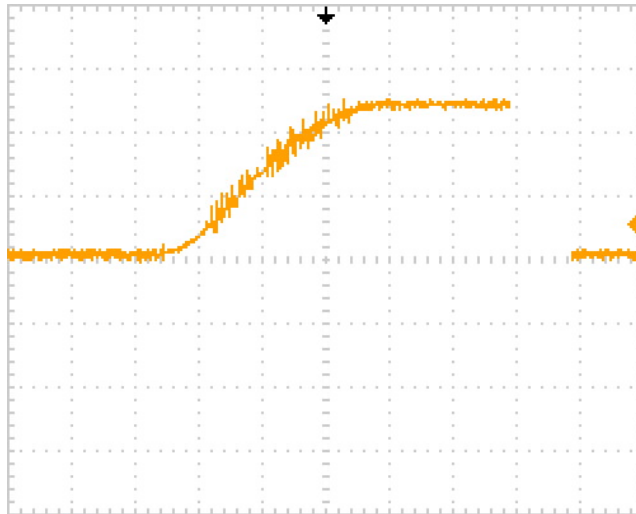


Figura 29. Respuesta al escalón del sistema real.

En la figura 30 se muestra la señal de control proporcionada por la DAQ, ante el mismo escalón aplicado para la figura anterior.

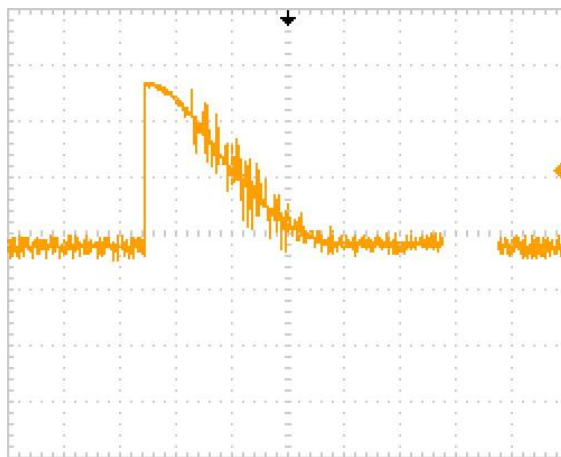


Figura 30. Señal de control obtenida para un escalón de entrada.

Se realizó una prueba de latencia de una tarea de tiempo real similar a la tarea de control programada, la programación de la tarea de tiempo real se muestran en la figura 31.

```

TIME1 = rt_get_time_ns();
/* insert here your periodic task */
comedi_data_read(it,subdev,3,range,aref, &entrada);
//salida = 0.64*entrada + 32730 ;
salida = 1.09*entrada;

comedi_data_write(it,1,0,range,aref, salida);

TIME2 = rt_get_time_ns();

//printf("\n");
printf("entrada: %d\n",entrada,"\n");
printf("salida: %d\n",salida,"\n");
printf("latencia: %f\n",TIME2 - TIME1,"\n");

```

Figura 31. Programación de una tarea de tiempo real similar a una de control.

Como se puede ver en la figura 31 se midió el tiempo del temporizador de RTAI¹⁰ antes y después de iniciar la tarea(TIME1 y TIME2), y se imprimió en pantalla el resultado de la diferencia de estos dos tiempos en la línea “latencia”.

Un screenshot de estos resultados se muestran en la figura 32.

entrada: 36111	entrada: 36099
salida: 39360	salida: 39347
latencia: 0.000025	latencia: 0.000033
entrada: 36098	entrada: 36137
salida: 39346	salida: 39389
latencia: 0.000032	latencia: 0.000023
entrada: 36069	entrada: 36107
salida: 39315	salida: 39356
latencia: 0.000029	latencia: 0.000033
entrada: 36092	entrada: 36088
salida: 39340	salida: 39335
latencia: 0.000034	latencia: 0.000022
entrada: 36085	entrada: 36100
salida: 39332	salida: 39349
latencia: 0.000030	latencia: 0.000023
entrada: 36105	entrada: 36104
salida: 39354	salida: 39353
latencia: 0.000023	latencia: 0.000024
entrada: 36123	entrada: 36096
salida: 39374	salida: 39344
latencia: 0.000024	latencia: 0.000029

Figura 32. Resultados de la prueba de latencia de la tarea de tiempo real.

¹⁰ Ver APIs ofrecidas por RTAI, en el manual de usuario de RTAI, sección “What time is it?”

La respuesta en el tiempo presentó un Error en Régimen Permanente de unos cuantos milivoltios, este error se pudo haber debido a ruido, que se puede apreciar claramente en las figuras 29 y 30.

El tiempo de asentamiento que presentó el sistema es de aproximadamente 2.5s, lo cual es aceptable para un control proporcional.

Tanto la señal entregada por el Sensor Magnético de Posición Absoluta(fig. 28) y la señal de control entregada por la DAQ(fig. 30), presentan mucho ruido eléctrico, esta es una de las fuentes de error en régimen permanente.

Un modelo empírico pudo haber sido de ayuda para saber que tan cercano era el modelo teórico desarrollado.

Como se muestra en la figura 32, el tiempo máximo de ejecución de la tarea fue de $34\mu\text{s}$, lo que concuerda con la operación de un RTOS en Hard-Real Time. Además según el resultado obtenido en la sección de [caracterización de la planta](#) la latencia máxima es más que suficiente para cumplir con el teorema de Nyquist, lo que demuestra que el sistema es capaz de muestrear y controlar la planta.

7 Conclusiones








- El Sistema Operativo Linux junto con el parche de software RTAI, ofrecen una plataforma fácil de programar, para desarrollar sistemas de control con todas las facilidades de un sistema operativo, además de que permite expandir las capacidades del sistema de control, como por ejemplo el desarrollo de una interfaz gráfica de usuario.
- RTAI puede utilizarse para sistemas de control de sistemas mecánicos, mas no para sistemas con una dinámica mas rápida, como por ejemplo la dinámica de un convertidor Buck.
- La implementación de controladores digitales en una plataforma de control basado en un computadora personal es mucho mas rápida y sencilla que su contraparte analógica.
- Un controlador implementado en una plataforma similar a la SAC es mas fácil de ajustar debido a que sus parámetros son números que pueden ser cambiados fácilmente en el programa.
- El ruido eléctrico en el sensor y en la señal de control son una fuente de error en régimen permanente.
- Un modelo empírico puede ser de ayuda para validar que tan cercano está al sistema real el modelado teórico.

8 Recomendaciones

- Para reducir el Error en Régimen Permanente se debería reducir el ruido presente en la señal generada por el Encoder Magnético Absoluto.
- Para lograr una caracterización de la planta mas exacta, esta no debe presentar mucho juego en los engranes, ya que esto presenta una fuente de errores en las mediciones.
- Realizar un modelo empírico de la planta puede ser un buen punto de referencia para el modelado teorico.
- Si se quiere mover los motores en sentido horario y antihorario se debe realizar un buck en puente completo.
- Una forma de reducir el efecto de los pulsos no producidos por la rotación del eje es utilizar un disco para el encoder incremental con menos cantidad de pulsos por revolución, de esta forma los espacios marcados en el disco no inducirán pulsos “falsos” ocasionados por el juego de los engranes.
- Todas las señales tanto de control como de retroalimentación deben estar en el mismo rango de tensiones, es decir si el Motor trabaja a 3V, el PWM debe ser de 3Vpp, y la señal del sensor debe ser acondicionada para que trabaje de 0V a 3V, este ajuste proporcionará una respuesta mas rápida del sistema, y el modelo teórico se acerca mas al modelo real, este criterio es conocido como rango teórico.

Bibliografía

- [1]  Park, J. Mackay, S. 2003. *Practical Data Acquisition for Instrumentation and Control Systems*. Elsevier.
- [2]  Dorf, R. Bishop, R. 2005. *Sistemas de Control Moderno*. Prentice Hall.
- [3]  OSHA Technical Manual(en línea). Consultado 10 Junio. 2012. Disponible en: http://www.osha.gov/dts/osta/otm/otm_iv/otm_iv_4.html#2
- [4]  Silberschatz. 2004. *Operating Systems Concepts*.
- [5]  Wulf, W. HYDRA: The Kernel of a Multiprocessor Operating System (en línea). Disponible en: www.cse.unsw.edu.au/~cs9242/11/papers/p337-wulf.pdf
- [6]  Walls, C. 2009. *Building a Real Time Operating Systems*. Estados Unidos (en línea). Consultado 9 Junio. 2012.
- [7]  González, V. Jara, L. 2009. *Kit Didáctico para Ingeniería de Control Utilizando Linux y RTAI* (en línea). Consultado 9 Junio. 2012. Disponible en: http://hedisc.ietec.org/proyectos/ucna1/resumen_ucna1.pdf
- [8]  National Instruments. 2012. *What is a Real-Time Operating System*. (en línea). Consultado 19 Junio. 2012. Disponible en: <http://www.ni.com/white-paper/3938/en>
- [9]  Sarolahti, P. 2001. *Real-Time Application Interface* (en línea). Consultado 19 junio. 2012. Disponible en: www.cse.iitb.ac.in/~cs684/RTATutors/rtai.pdf .
- [10]  Halvorsen Hans-Peter. 2011. *Data Acquisition in LabVIEW* (en línea). Consultado 19 Junio. 2012. Disponible en: <http://www.scribd.com/doc/65487668/Data-Acquisition-in-LabVIEW>
- [11]  Schleeff, D. 2012. *The Control and Measurement Device Interface Handbook for Comedilib 0.10.0* (en línea). Disponible en: <http://www.comedi.org/doc/index.html>
- [12]  Saad, M. 2007. *Buck Converter Design Issues* (en línea). Consultado 8 Junio. 2012.
- [13]  Guiggiani, A. 2012. *Set up a machine for real-time remote control using RTAI, Scicoslab and RTAI-XML*(en línea). Disponible en: <http://www.rtaixml.net/realtime-suite>

- [14]  Schlee, D. 2012. The Control and Measurement Device Interface Handbook for Comedilib 0.10.0 (en línea). Disponible en: <http://www.comedi.org/doc/index.html>
- [15]  Racciu, G. Mantegazza, P. 2006. RTAI User Manual.
- [16]  Coughling, R. Driscoll, F. 1999. Amplificadores Operacionales y Circuitos Integrados Lineales. Prentice Hall.
- [17]  Anritter, F. Maurer, P. Reger, J. Flatness Based Control of a Buck-Converter Driven DC Motor.
- [18]  Kuo. B. Sistemas de Control Automático. Prentice Hall.
- [19]  DCMS. DC Motor System User Manual (en línea). 2011. Zeltom LLC. Disponible en: http://zeltom.com/documents/dcms_um_13.pdf.
- [20]  Kuo. B. Sistemas de Control Automático. Prentice Hall.

Anexos

Anexo1: Instalación de RTAI en Ubuntu 10.04 LTS

TUTORIAL RealTime Suite v1.2

Set up a machine for real-time remote control using

RTAI, Scicoslab and RTAI-XML

Alberto Guiggiani

guiggiani@gmail.com

Mar 10, 2012

Introduction

This tutorial is focused on providing you all the information needed to quickly build a working control machine for real-time signal acquisition and processing.

It's completely based on open-source software, all included in the archive provided.

Most of the procedures described in the first half of this article come from the work of Roberto Bucher (University of Applied Sciences of Southern Switzerland), Simone Mannori (INRIA, Rocquencourt) and Thomas Netter (University of Zurich), while for part that covers the remote control credit goes to the work of Marco Romagnoli, Michele Basso (Università degli Studi di Firenze) and Massimo Vassalli (Istituto di Biofisica, CNR, Genova).

I edited the package sources to ensure compatibility with each other's version and the operating system suggested, and to possibly avoid compiling errors and malfunctions while following the steps of this tutorial in the exact order. Considering this, I advise you to get all software from the realsuitesuite archive instead of their official versions available on the web.

Part 1 - The software chain

Here is a brief description of all the software you're going to use.

- **Comedi:** www.comedi.org

A set of drivers that allows your machine to interface with a great number of data acquisition (DAQ) boards.

- **RTAI:** www.rtai.org

Real Time Application Interface - an extension to Linux Kernel that allows the execution of processes with hard real-time features.

- **Scicoslab:** www.scicoslab.org

A popular scientific computation environment - provides a CACSD (Control Applications Computer Aided Control System Design) GUI for modeling and control.

- **RTAI-Lib**

Adds a toolbox to Scicoslab to interface with DAQ's ports, as well as compiling real-time executables.

- **QRTAILab:** qrtailab.sourceforge.net

QRTAILab can connect to a running target and monitor scopes, meters and LEDs, as well as changing model parameters on-the-fly.

- **RTAI-XML:** www.rtaixml.net

This software is composed of two parts: a server running in background on your real-time control machine, and a GUI client that allows you to monitor target and control its parameters from a remote machine through the TCP/IP protocol.

Step 1 - Prepare your machine

- Perform a clean installation of Ubuntu 10.04 x86 and apply all available updates.

- From now on, all commands that follow a white dot have to be typed in a terminal window (follow the rule that for each dot corresponds one command on a single line).

- Get the following packages:

- sudo apt-get install cvs subversion build-essential libtool automake libncurses5-dev bison flex libboost-dev libboost-program-options-dev libgsl0-dev gfortran sablotron tcl8.5-dev tk8.5-dev xaw3dg-dev libpvm3 pvm-dev libgtkhtml2-dev libvte-dev ocaml-native-compilers libqt4-dev libqwt5-qt4-dev xfonts-100dpi xfonts-75dpi ssh

- If you don't already have it get realsuite archive, then extract it:

- sudo wget http://sourceforge.net/projects/rtaixml/files/realsuite/1.2/realsuite-1.2.tar.gz/download

- sudo mv download /usr/src/realsuite-1.2.tar.gz

- cd /usr/src

- sudo tar -xvzf realsuite-1.2.tar.gz

- Ensure that /usr/src folder contains at least the following

subfolders: 'boot', 'comedi', 'comedi_calibrate', 'comedilib', 'comedi-nonfree-firmware', 'ede', 'efltk', 'jrtailab', 'kernel', 'qrtailab', 'rtai', 'rtaixml', 'scicoslab'.

- Install EFLTK library:

- cd /usr/src/efltk

- sudo autoconf

- sudo ./configure --disable-mysql --disable-unixODBC

- sudo ./emake

- sudo ./emake install

- sudo gedit /etc/ld.so.conf

Add a line with the path /usr/local/lib and save.

- sudo /sbin/ldconfig

Step 2 - RTAI Patched Kernel

- Install patched kernel:

- cd /usr/src

- sudo gdebi linux-headers-rtai_all.deb

- sudo gdebi linux-image-rtai.deb

- Check contents of src folder:

- cd /usr/src

- ls

- Find version of rtai kernel folder in the form

linux-headers-x.x.x-rtai (e.g. linux-headers-2.6.32-122-rtai)

and make symbolic link (remember to replace version number):

- `sudo ln -s linux-headers-2.6.32-122-rtai linux`
- Reboot in the new Kernel:
- `sudo reboot`
- After reboot, check if patched kernel is running:
- `uname -r`

Step 3 - First RTAI Installation

- Install Comedilib:
 - `cd /usr/src/comedilib`
 - `sudo sh autogen.sh`
 - `sudo ./configure --sysconfdir=/etc`
 - `sudo make`
 - `sudo make install`
 - `sudo make dev`
- Check the number of CPUs available on your machine:
 - `cat /proc/cpuinfo | grep processor | wc -l`
- Enter RTAI compiling options:
 - `cd /usr/src/rtai`
 - `sudo make menuconfig`
- Under Machine, set the number of CPUs that you previously found.
- Exit Configuration, choosing 'Yes' when prompted to apply changes.
- Compile and install RTAI:
 - `sudo make`
 - `sudo make install`
 - `sudo sed -i 's/(PATH=\\")\\1\\usr\\realtime\\bin:/' /etc/`

environment

- Reboot the system.
- After reboot, set RTAI kernel modules to be loaded at system startup:
 - `cd /usr/src/boot`
 - `sudo cp rtailab /etc/init.d/`
 - `sudo chmod a+x /etc/init.d/rtailab`

- sudo update-rc.d rtailab defaults
- Now load RTAI kernel modules without Comedi:
- sudo /etc/init.d/rtailab preload

Step 4 - Comedi Installation

- Create directory:
 - sudo mkdir /usr/local/include/linux
- Install Comedi:
 - cd /usr/src/comedi
 - sudo sh autogen.sh
 - sudo ./configure --with-linuxdir=/usr/src/linux --with-rtaidir=/usr/realtime
 - sudo make
 - sudo make install
 - sudo depmod -a
 - sudo make dev
 - sudo ldconfig
- Install Comedi Calibrate:
 - cd /usr/src/comedi_calibrate
 - sudo autoreconf -i -B m4
 - sudo ./configure
 - sudo make
 - sudo make install

Step 5 - Second RTAI Installation

- Copy and link the following files:
 - sudo cp /usr/src/comedi/include/linux/comedi.h /usr/local/include/
 - sudo cp /usr/src/comedi/include/linux/comedilib.h /usr/local/include/
 - sudo ln -s /usr/local/include/comedi.h /usr/local/include/linux/comedi.h
 - sudo ln -s /usr/local/include/comedilib.h /usr/local/include/linux/comedilib.h
- Enter RTAI compiling options:
 - cd /usr/src/rtai

- sudo make menuconfig
- Under Addons, select 'Real Time COMEDI support in user space' (by pressing spacebar).
- Under RTAI Lab, select 'RTAI Lab'.
- Exit and save configuration.
- Install RTAI:
 - sudo make
 - sudo make install
 - sudo cp /usr/src/comedilib/include/comedilib.h /usr/local/include/
- Navigate with your web browser to <http://www.comedi.org/hardware.html> and find your DAQ hardware in the list; take note of name under "Driver" column.
- Edit rtailab startup file with driver for your DAQ board:
 - sudo gedit /etc/init.d/rtailab
- Replace driver on lines 33 and 55 with the one you found on Comedi website. Save and close.
- Reboot system.

Step 6 - Scicoslab Installation

- Navigate with File Browser to /usr/src/scicoslab
- Right-click on 'scicoslab.deb' and select 'Open with GDebi'.
- Install package.
- Repeat for 'tkdnd1_1.0-1_i386.lucid.deb'.
- Add RTAI toolbox and palette to Scicoslab:
 - sudo ln -s /usr/realtime/bin/rtai-config /usr/local/bin
 - cd /usr/src/rtai/rtai-lab/scicoslab/macros/
 - sudo make install
 - make user
- Navigate with File browser to /usr/lib and check scicoslab-gtk folder name. Then make symbolic link accordingly:
 - cd /usr/local/bin
 - sudo ln -s /usr/lib/scicoslab-gtk-4.x/bin/scilab scilab

- Start Scicoslab as root and check the presence of toolboxes->RTAI in the upper menu:
- `sudo scicoslab`

Step 7 - QRTAILab Installation

- Install QRTAILab:
- `cd /usr/src/qrtailab/qrtailab`
- `sudo qmake-qt4`
- `sudo make`
- `sudo make install`

Step 8 - RTAI-XML Installation

- Install RTAI-XML (pay attention to terminal outputs, you'll get useful informations on how to use it):
- `cd /usr/src/rtaixml`
- `sudo make all`
- `sudo make install`
- Set RTAI-XML modules to be loaded at system startup:
- `sudo cp /usr/src/boot/rtaixml /etc/init.d/`
- `sudo chmod a+x /etc/init.d/rtaixml`
- `sudo update-rc.d rtaixml defaults`
- Start RTAI-XML modules:
- `sudo /etc/init.d/rtaixml start`

Step 9 - Set up jRTAI-Lab for remote control

- The following instructions are for Ubuntu 10.04, though you can easily run jRTAI-Lab on any OS that supports Sun Java Virtual Machine 6.
- Install JVM6:
- `sudo add-apt-repository "deb http://archive.canonical.com/lucid partner"`
- `sudo apt-get update`
- `sudo apt-get install sun-java6-jre`
- Allow execution of jRTAI-Lab and run it:
- `sudo chmod a+x /usr/src/jrtailab/jRtaiLab.jar`

- `java -jar /usr/src/jrtailab/jRtaiLab.jar`

Congratulations, you're done!

Part 3 - Your first real time target

In this last part of the tutorial I'll show the steps needed to compile a simple target and remotely interact with it.

The objective is to display a sine wave and control its amplitude.

Step 1 - Build the model using Scicos

The first thing to do is to use Scicoslab modeling GUI - Scicos - to create the model. Keep in mind the following figure when placing blocks.

- Create a subfolder in your home folder named test
- Open Scicoslab.
- In upper bar, toolboxes -> RTAI. Check for "Scicos-RTAI Ready" output.
- Type scicos in command window to open modeling GUI.
- In the new window, Palette -> Pal Tree to open the list of available blocks.
- Expand Sources and grab the red clock. Place it in the upper part of your model.
- Double-click the clock. From here you can set operating frequency. Type 1/1000 as Period (1KHz)
- In Palettes windows, expand Linear and drag gain block to the model, directly below the clock.
- Right-click on the gain block you just placed and select Block Properties -> Identification.
Type gain.
- In Palettes windows, expand RTAI-Lib and place a Sine block and a Scope block respectively to the left and right of gain block.
- Double-click the Sine block and set Frequency to 50.
- Now connect blocks as shown in figure. (TIP: to branch a line, just right-click it and select Link).

Step 2 - Compile the real-time executable

- In scicos window, drag a selection that covers all the model except for the clock.
- In upper bar, Diagram -> Region to Super Block. It's mandatory that all your model is contained in a single Super Block with only Clock as input.
- Highlight the Super Block and, in upper bar, RTAI -> Set Target. Confirm with OK.
- The super block will automatically expand in a new window. Go back to the main scicos windows, highlight again the Super Block and, in upper bar, RTAI -> RTAI CodeGen.
- Under New block's name you can chose the target name. For this example, name it testsine
- Under Created files Path, enter /home/yourusername/test
- Confirm with OK and check Scicoslab window for compiling results. If you see the output "### Created executable: testsine ###" then your target is ready to run.

Step 3 - Test the target with QRTAILab

- In terminal, type
 - qrtailab &
 - /home/yourusername/testsine
- Check is "TARGET STARTS." is prompted as output in terminal.
- Switch to QRTAILab window.
- In upper bar, Target -> Connect. You should see in the lower left-hand corner of the window the message "Target: testsine".
- In upper bar, View -> Scope. Check "Show/Hide" option and adjust "sec/div" to get a nicer visualization (i.e. 0.05). Then close manager.
- Now we try to control wave amplitude on-the-fly. In upper bar, View -> Parameter.
- Double click on gain, then select Block Parameters tab. Try different values and see changes in the scope
- Now go back to QRTAILab main window. In upper bar, Target -> Stop. Then close QRTAILab.

Step 4 - Remotely control your target with jRTAI-Lab

- The first things to do are to properly configure server machine. In terminal:
 - `cd /usr/realtime/rtaixml/scripts`
 - `sudo cp TEST STEST`
 - `sudo gedit STEST`
- This is the configuration script for rtaixml. You have to configure a copy of it for each target you want to remotely control. NOTE: the file name shouldn't exceed six characters long.
- Under “#Name of the target executable”, you need `target=testsine`.
- Under “#Absolute path”, `dir=/home/yourusername`
- Set “Final Time” to 1000
- Remove the “#” before “`wait=true`” and set it to false.
- RTAI-XML server is ready. Move on the client machine.
- Start jRTAI-Lab (see the end of Part 2 of this document) and click on the button that pops up.
 - In upper bar, File -> Connect
 - Edit IP of server machine, leave “Server Port” to default and in “Target ID” insert the script name you created for the target (in this example, STEST).
 - In upper bar, View -> Signal. Click on SCOPE then check Plot.
 - In upper bar, View -> Parameters. Here you can live adjust model parameters as you did in QRTAILab (NOTE: remember to press ENTER in the parameter value, “Set” button alone isn't enough).

Anexo 2: Modelado Matemático del Buck y la planta.

En la figura A2.1 se muestra el modelado del Buck junto con uno de los Motores CD, en el modelado matemático se tomaron en cuenta la fricción viscosa y el momento de inercia tanto de la caja de engranes como del motor.

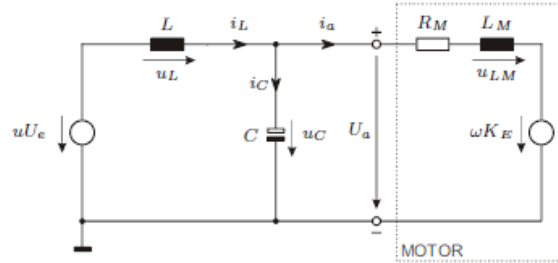


Figura A2.1. Modelo del Buck y de un Motor.

Las ecuaciones de esta que se obtienen con base en el modelo de la Figura A2.1 son:

$$\frac{di_L(t)}{dt} = -\frac{1}{L}u_C(t) + \frac{E}{L}U_e(t)$$

$$\frac{du_C(t)}{dt} = \frac{1}{C}i_L(t) - \frac{1}{C}i_m(t)$$

$$\frac{di_m(t)}{dt} = \frac{1}{L_m}u_C(t) - \frac{R_m}{L_m}i_m(t) - \frac{K_e}{L_m}\omega(t)$$

$$\frac{d\omega(t)}{dt} = \frac{k_t}{J}i_m(t) - \frac{B}{J}\omega(t)$$

De las ecuaciones de estado se obtienen las siguientes matrices:

$$A = \begin{bmatrix} 0 & \frac{-1}{L} & 0 & 0 \\ \frac{1}{C} & 0 & \frac{-1}{C} & 0 \\ 0 & \frac{1}{L_m} & \frac{-R_m}{L_m} & \frac{-k_e}{L_m} \\ 0 & 0 & \frac{k_T}{J} & \frac{-B}{J} \end{bmatrix} \quad B = \begin{bmatrix} \frac{E}{L} \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad C = [0 \quad 1 \quad 0 \quad 0]$$

Y la función de transferencia es:

$$Y(s) =$$

$$\frac{E[L_m \cdot J \cdot s^2 + (R_m \cdot J + B \cdot L_m) \cdot s + B \cdot R_m + k_e \cdot k_T]}{(L_m \cdot C \cdot J \cdot L) \cdot s^4 + (R_m \cdot C \cdot J \cdot L + B \cdot C \cdot L_m \cdot L) \cdot s^3 + (J \cdot L + B \cdot C \cdot R_m \cdot L + C \cdot k_e \cdot k_T \cdot L + L_m \cdot J) \cdot s^2 + (B \cdot L + R_m \cdot J + B \cdot L_m) \cdot s + B \cdot R_m + k_e \cdot k_T}$$

Anexo 3: Controlador implementado utilizando las capacidades de RTAI

```
/*RTAI headers*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/poll.h>
#include <sys/mman.h>
#include <signal.h>
#include <sys/io.h>
#include <rtai_lxrt.h>

/*Comedi headers*/
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <signal.h>
#include <math.h>
#include <rtai_comedi.h>

/*KBHIT headers*/
#include <termios.h>
#include <fcntl.h>

/*****
*****/
/***** Variables del controlador *****/
/*****
*****/

int V_ref_1, V_ref_2; //señales de control para los dos motores
int m_pos_set_1, m_pos_set_2; //set point al que se quiere que lleguen los motores
int e_m_pos_1, e_m_pos_2; //señal de error de

/*****
*****/
/***** Variables de Comedi *****/
/*****
*****/

int subdev = 0; /* change this to your input subdevice */
int chan = 0; /* change this to your channel */
unsigned int range = 0; /* more on this later */
int aref = AREF_GROUND; /* more on this later */

/*****
*****/
/***** Variables de RTAI Task *****/
/*****
*****/

int ns = 1000000;
static volatile int endex;
static RTIME rt_BaseRateTick;
```

```

        /*****
        /*****  KBHIT  *****/
        /*****/
int kbhit(void)
{
    struct termios oldt, newt;
    int ch;
    int oldf;

    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
    fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);

    ch = getchar();

    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    fcntl(STDIN_FILENO, F_SETFL, oldf);

    if(ch != EOF)
    {
        ungetc(ch, stdin);
        return 1;
    }

    return 0;
}

```

```

        /*****
        /*****  MAIN  *****/
        /*****/

```

```

static int rt_Main()
{
    /*comedi*/
    comedi_t *it;                //Puntero al driver de NI PCI 6221
    lsampl_t M_pos_1;           //Variable de posicion de eje de Motor 1
    lsampl_t M_pos_2;           //Variable de posicion de eje de Motor 2

    it=comedi_open("/dev/comedi0"); //abre el driver para la tarjeta
    /*comedi*/
}

```

```

/*RTAI Task*/
unsigned int rt_BaseTaskPeriod;
double Tsamp = 0.000001;
static RTIME t0;
double TIME1,TIME2;
static RT_TASK *rt_MainTask;
static int out = 0;

if (!(rt_MainTask = rt_task_init_schmod(nam2num("TMN"), 0, 0, 0, SCHED_FIFO, 0xFF))) {
    fprintf(stderr,"Cannot init rt_MainTask.\n");
    return 1;
}

rt_BaseTaskPeriod = (unsigned int) (1e9*Tsamp);
rt_BaseRateTick = nano2count(rt_BaseTaskPeriod);

rt_BaseTaskPeriod = (unsigned int) (1e9*Tsamp);
rt_BaseRateTick = nano2count(rt_BaseTaskPeriod);

iopl(3);
rt_task_use_fpu(rt_MainTask, 1);
mlockall(MCL_CURRENT | MCL_FUTURE);

rt_make_hard_real_time();

t0 = rt_get_cpu_time_ns();

rt_task_make_periodic(rt_MainTask, rt_get_time() + rt_BaseRateTick, rt_BaseRateTick);
while (!endx) {
    rt_task_wait_period();

TIME1 = (rt_get_cpu_time_ns() - t0)*1.0E-9;

/*RTAI Task*/

/* ***** */
/* Tarea Periodica */
/* ***** */

```

```

/*****
/***** Controlador *****/
/*****/

comedi_data_read(it,subdev,3,range,aref, &M_pos_1); //Lectura de la entrada analogica 3
comedi_data_read(it,subdev,4,range,aref, &M_pos_2); //Lectura de la entrada analogica 4

//CONTROL DE POSICION
m_pos_set_1 = 47000; //Punto al que debe llegar el eje del Motor 1, 32800(0V) < m_pos_set1 < 48040(5V)
m_pos_set_2 = 47000; ///Punto al que debe llegar el eje del Motor 2, 32800(0V) < m_pos_set1 < 48040(5V)

e_m_pos_1 = ( m_pos_set_1 - (M_pos_1)); //Error de posicion
e_m_pos_2 = ( m_pos_set_2 - (M_pos_2)); //Error de posicion

V_ref_1 = k * e_m_pos_1 + 32750 ;
V_ref_2 = k * e_m_pos_2 + 32750 ;
//Implementacion del control proporcional
//el numero 32750 es el valor de 0V para la tarjeta, ya que esta trabaja con tensiones negativas

/* Imprime en pantalla la posicion del eje de cada Motor*/
printf("\n");
printf("M_pos_1: %d\n",M_pos_1,"\n");
printf("M_pos_2: %d\n",M_pos_2,"\n");

/* Escritura de las senales de control */
comedi_data_write(it,1,0,range,aref, V_ref_2); //escribe en salida 22
comedi_data_write(it,1,1,range,aref, V_ref_1); //escribe en salida 21

/* Escritura de las senales de control */
comedi_data_write(it,1,0,range,aref, V_ref_2); //escribe en salida 22
comedi_data_write(it,1,1,range,aref, V_ref_1); //escribe en salida 21

/***** PARA ACABAR EL PROCESO*****/
if(kbhit()) {
stop_rt_timer(); //Detiene el temporizador
rt_make_soft_real_time(); //Convierte la tarea en una de tiempo real flexible
rt_task_delete(rt_MainTask); //Elimina la tarea
}

if(out) out = 0;
else out = 1;
outb(out,0x378);
}
}

```

```
/*  
***** Funciones de RTAI *****  
*/
```

```
static void endme(int dummy)  
{  
    signal(SIGINT, endme);  
    signal(SIGTERM, endme);  
    endex = 1;  
}
```

```
int main(int argc, char *argv[])  
{  
    signal(SIGINT, endme);  
    signal(SIGTERM, endme);  
  
    return rt_Main();  
}
```