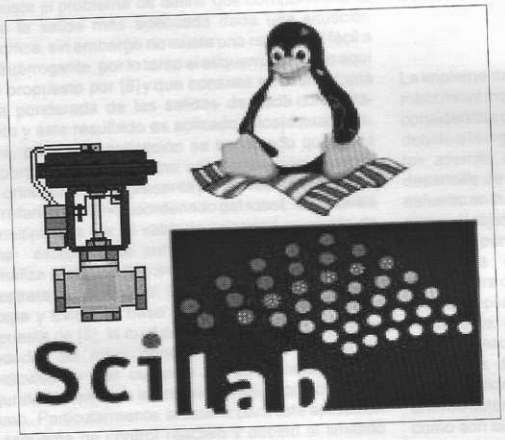
APLICACIONES DE LINUX PARA SIMULACIÓN Y CONTROL DE PROCESOS

Software GNU: Scilab y Scicos



- Ingeniero **Edinson Franco Mejía**, Ingeniero Electricista (1992) y Magíster en Automática (1994) de la Universidad del Valle. Profesor Titular de la Escuela de Ingeniería Eléctrica y Electrónica de la Universidad del Valle. Actualmente es Doctorado en la misma institución.
- **Carlos A. Ramos**
  Ingeniero Electrónico
  Universidad del Valle.
- **Angela María Maya**
  Ingeniera Electrónica
  Universidad del Valle

Grupo de Investigación en Control Industrial, GICI.

RESUMEN

En este artículo se presentan los resultados de la aplicación del Scilab y Scicos bajo Linux para aplicaciones de control con respuesta en tiempo real. Se inicia con una breve descripción del Software, se continúa con una explicación de las herramientas de hardware y software necesarios para correr la aplicación, y se finaliza mostrando un controlador RST para la planta de control de presión.

PALABRAS CLAVES

Software Educativo, Control RST.

ABSTRACT

This paper shows the results of the Real time control applications included in the Scilab and Scicos on Linux operating system. A brief software review and a description of the hardware and software required to execute the toolbox is presented. The paper ends with an example showing the implementation of a RST controller for the PCT14 Feedback, pressure process.

KEYWORDS:

Educational Software, RST Control.

* Recibido : Diciembre 2002
* Aceptado: Septiembre 2003

1. INTRODUCCIÓN

Usando la tarjeta de adquisición de datos TIM[1], la tarjeta de adquisición de la National Instruments (NIDAQ 16DH), Scilab y Scicos, se realizó el control de la respuesta de un proceso en tiempo real, tomando y escribiendo datos por medio de la tarjeta, monitoreando y controlando por medio del Scicos y aplicando las herramientas que posee Scilab a las señales obtenidas.

Ya que en este trabajo el procesamiento de la información es secuencial y que las señales tienen respuestas muy lentas en comparación con las velocidades de procesamiento, se utiliza aquí el término "*respuesta en tiempo real*" que no debe confundirse con los conceptos de manejo de tareas en tiempo real, mas conocido hoy día como "*sistemas en tiempo real*".

El objetivo central esta enfocado a mostrar la potencialidad de la herramienta para el control de procesos mas que a obtener una buena estrategia de control RST [5].

2. DESCRIPCIÓN DEL SCILAB Y DEL SCICOS

Scilab es un paquete matemático desarrollado por INRIA. [2] Scilab tiene compatibilidad para programación en C y fortran, posee manipulación de matrices, calculo simbolico y muchas otras aplicaciones matemáticas. La filosofía general de Scilab se basa en proporcionar un entorno computacional para: tener tipos de datos flexibles con sintáxis natural y fácil de usar, disponer de una razonable cantidad de primitivas para realizar gran variedad de cálculos, tener un sistema de desarrollo donde adicionar nuevas primitivas sea fácil (una herramienta de desarrollo fácil de usar distribuida con SCILAB es INTERSCI, desarrollada para crear primitivas y interfaces, adicionar módulos de Fortran y C a SCILAB, etc.), poseer una ayuda en línea con el comando help, y soportar librerías de desarrollo (toolboxes) de funciones para aplicaciones específicas, tales como: control lineal, control no-lineal, procesamiento de señales, etc[1].

Scilab está estructurado en directorios muy específicos tales como: bin, demos, examples, doc, geci, pvm3, imp, macros, man, maple, routines, intersci, scripts, tests, util, xless y xmetanet. Scilab permite al igual que el matlab manipular matrices y vectores, los tipos de datos soportados por Scilab más usados son:

Constantes especiales

%i (número imaginario), %pi (número PI = 3.141592...), %e (número e exponencial), %eps (número tal que 1+%eps = 1), %s (variable simbólica, s=poly(0,'s')), %z (variable simbólica, z=poly(0,'z')), %t (valor booleano verdadero), %f (valor booleano falso).

Matrices y vectores

Definición de un vector, ejemplo: --> v=[1 5 6]
Definición de matrices, ejemplo: --> A=[2 1 4;5 -8 2], ejemplo: --> b=ones(2,3)

2.1. Generando funciones y sistemas en Scicos

El Scilab y el Scicos, permiten trabajar modelos matemáticos de sistemas representados en función de transferencia y/o en espacio de estados, a continuación un ejemplo de la definición de un sistema en función de transferencia; sea el sistema descrito por la ecuación 1, lo primero que se debe hacer es definir la variable s, existen dos maneras: s=poly(0,'s') ó s=%s. Posteriormente se debe escribir la función de transferencia G(s) como: àGs=(s+2)/(s*(s+3)); finalmente, se debe formar un sistema lineal con la función escrita para que el Scilab la interprete:à Gsl=syslin('c',Gs), donde el primer argumento de la función syslin es un carácter "c" que indica que el sistema es de tiempo continuo, para sistemas de tiempo discreto se usa el carácter "d".

$$G(s) = \frac{s+2}{s(s+3)} \qquad (1)$$

Una forma abreviada de hacer lo mismo es digitando en una línea:
--> s=%s, Gsl=('c', (s+2)/(s*(s+3)));.

Para la simulación de sistemas están disponibles las funciones "csim" para simulación continua y "dsimul" para simulación de sistemas discretos[2].

2.2 Opciones de graficación

El Scilab posee comandos para graficación al igual que el Matlab, para ello emplea las funciones plot y plot2d, permite gráficar varias señales en un mismo cuadro y también tener varios cuadros de gráficas en una sola ventana. Además, es posible gráficar en tres dimensiones como se ilustra en la Figura 1.

2.3 Descripción del SCICOS

El Scicos es un software para simulación de sistemas orientado a objetos, esta estructurado por paletas (o menús) que contienen los bloques básicos para estructurar diagramas de bloques de sistemas de control, se puede llamar la herramienta desde Scilab a través del comando: scicos(). En la sección 4 se puede observar la ventana típica con una aplicación.

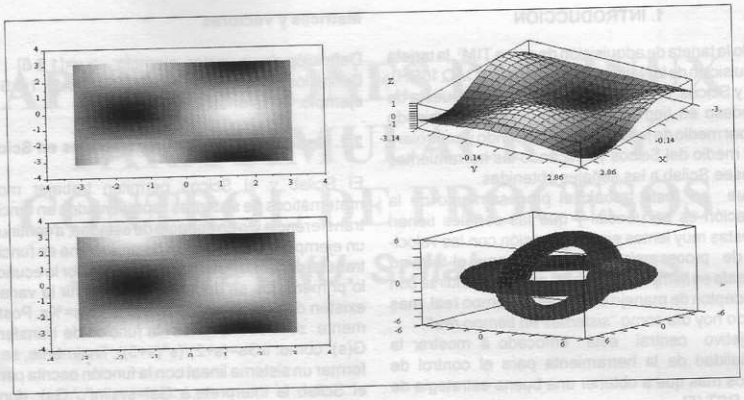


Figura 1. *Múltiples Gráficos en una ventana*

3. IMPLEMENTACION DE LOS CONTROLADORES PARA LA TARJETA DE ADQUISICIÓN DE DATOS Y LA CREACIÓN DE SUS BLOQUES EN SCICOS

La tarjeta TIM posee 8 canales de entradas análogas de 8 bits (0-5 Volts), 2 salidas análogas (0-5V) y 3 canales de salida digital de 8 bits. Para el manejo de las tarjetas de adquisición de datos se construyeron librerías en C bajo Linux, usando las librerías unistd.lib y asm/io.lib para acceder a los puertos necesarios. Antes de manejar datos por los puertos debe solicitarse permiso para acceder a una dirección hexadecimal de los puertos de entrada - salida. Las librerías desarrolladas son de enlace dinámico, reservando 16 o 32 puertos para intercambio de datos, dependiendo de la tarjeta a usar. Las funciones escritas permiten recibir y enviar palabras y bytes (representación de voltajes) desde y hacia los puertos[3].

Las librerías desarrolladas de enlace dinámico permiten acceder a los recursos de la tarjeta *simultáneamente* por aplicaciones no relacionadas y con ejecución independiente; es decir, gracias al desarrollo de la librería de acceso en forma no estática se puede leer un dato en la entrada análoga 0, escribir 1AH en el puerto digital 2 y poner 3.4 volts en la salida análoga 1, todo en un mismo instante de tiempo.

Los conversores digital a análogo (DAC) de ambas tarjetas tienen un retardo estimado de 10ms, así que para acceder al dato convertido se debe esperar 15ms. El conversor análogo a digital (ADC) de la tarjeta UV tiene un retardo de 100ms, y el ADC de la tarjeta NI tiene un retardo de 50ms.

3.1 Descripción de las funciones desarrolladas en C.

Los nombres de las librerías dinámica son Uvadq.a y national. a; las cabeceras de las funciones se encuentran en Uvadq.h o national.h y su estructura se puede ver a continuación:

- Inicia la tarjeta de adquisición en el puerto base: "puerto_base", generalmente 0x300, retorna 0 en caso de éxito o retorna el valor del error en caso de fallo (este valor esta definido en errno.h y puede conocerce el mensaje de error con la rutina perror()).

  int Inic_UV_adq(int puerto_base);
  int Inic_NI_adq(int puerto_base);

- Lee un dato análogo de un canal de entrada (0,1,.. 7) y asigna a la salida el valor del voltaje leído en volts, retorna el voltaje en caso de éxito y -1 en caso de error. Para el caso de la NI, las entradas van desde 0 hasta 7 para voltajes bipolares o desde 0 hasta 15 para voltajes unipolares; la polaridad se define con un bit (0 para unipolar, 1 para bipolar).

  float AI_VRead(int canal_ai);
  loat AI_VRead(int canal_ai, polaridad);

- Lee "num_canales" canales análogos de entrada definidos por el array "canales_ai" y coloca los valores de voltajes en el array "voltajes", retorna 0 en caso de éxito y en caso de error retorna los números de los errores.
  int AI_VRead_Scan(int num_canales,int canales_ai[], float voltajes[]);

- Escribe en el canal análogo de salida "canal_ao" (0 ó 1) el voltaje definido por "voltaje", retorna 1 en caso de éxito y -1 en caso de error; para la NI tiene una polaridad la cual es automáticamente gestionada por el sistema de acceso al hardware.

  int AO_VWrite(int canal_ao, float voltaje);
  int AO_VWrite(int canal_ao, float voltaje);

- Lee un bit específico (0,1,2,... ó 7) de un canal digital de entrada/salida (0, 1 ó 2) de 8 bits determinado por "bit" en el canal "canal_d" y lo pone en "inbit"; en caso de error retorna -1 y en caso de éxito retorna 1.

  int DIG_In_Line(int canal_d, int bit, int inbit);
  int ABDIO_In_Line(int canal_d, int bit, int inbit);

- Lee un canal digital (0,1 ó 2) de 8 bits determinado por "canal_d", y coloca el valor en "inbyte"; en caso de error retorna -1 y en caso de éxito retorna 1.

  int DIG_In_Port(int canal_d, int inbyte);
  int ABDIO_In_Port(int canal_d, int inbyte);

- Escribe en un canal digital (0,1 ó 2) de 8 bits determinado por "canal_d" en el bit numero "bit" (0,1,... 7) el valor "outbit" ; retorna 0 para éxito y -1 para error.

  int DIG_Out_Line(int canal_d, int bit, int outbit);
  int ABDIO_Out_Line(int canal_d, int bit, int outbit);

- Escribe en un canal digital (0,1 ó 2) de 8 bits determinado por "canal_d" el byte que se encuentra en "Outbit"; retorna 0 para éxito y -1 para error.

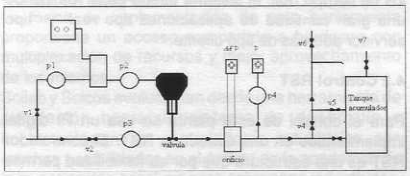
  int DIG_Out_Port(int canal_d, int byte);
  int ABDIO_Out_Port(int canal_d, int byte);

3.2 Interfaz de las funciones desarrolladas en C y Scilab

Se debe realizar una interfaz con funciones en C, de acceso a la tarjeta. Podría emplearse directamente la librería, pero para la comunicación con Scilab el paso de parámetros a la función debe ser por referencia y no por valor; esto limitaría el uso de la librería en Scilab. Con el modelo de diseño usado se puede acceder a la tarjeta en tiempo real y permitir que sean usados recursos libres de la tarjeta en aplicaciones externas al control que se está realizando, tales como mover una cámara CCD o mover un motor [1][3].

Existen cuatro funciones básicas de interfaz:

- Interfaz para la salida digital, los parámetros son el canal, el byte y el estado y son dados por referencia; como Scilab solo acepta parámetros pasados por valor, se hace un llamado a la función de la librería de la tarjeta que escribe los datos digitales por el puerto.
- Interfaz de lectura de datos análogos.
- Función para escribir un dato análogo enviando por valor el canal y el voltaje a la función de la librería que escribe la información en la tarjeta de adquisición.
- Función para leer datos digitales.



Posteriormente las funciones son compiladas, enlazadas y enmascaradas en un archivo con extensión "sce".

A continuación los archivos se cargan en Scilab a través de la función getf.

Para enlazar el anterior código o funciones como un objeto en Scicos, se construye el bloque con la función "Scifunc" que permite cargar una función del Scilab, luego se llama la función del Scilab, y se envían los parámetros exigidos por la función. Este bloque se estimula con un Oscilador, para obligar al Scicos a llamar continuamente a la librería dinámica, y así tener un flujo constante de datos, solo contado por la frecuencia del tiempo de muestreo que se desee. Por ejemplo para la escritura análoga se tienen dos parámetros, Y1 = UVoutanal (U1,U2).
Se puede emular una entrada análoga, haciendo el tiempo de muestreo tan cercano a cero como se quiera, hasta llegar al límite del hardware, que en el caso de la TIM es de al menos 100ms, (debido al tiempo de conversión del A/D usado en esta tarjeta ADC0809 de National Semiconductor).

4. CONTROLADOR EN TIEMPO REAL USANDO SCICOS

4.1 Entorno tiempo real

Para implementar controladores industriales es necesario contar con un entorno en tiempo real; en Scilab se dispone de un Toolbox de tiempo real [7]desarrollado por Anders Blomdell [7], (Departament of Automatic Control, Lund institute of Tecnology: anders. blomdell@ control. lth. se), Referencias basado en una semi-interrupción del procesador utilizando la cantidad de tics (ciclos del reloj del

procesador); en consecuencia, la resolución del tiempo real está ligada a la velocidad del reloj del procesador. Usando una versión de *linux* es posible tener tiempo real siempre y cuando el procesador no tenga mucha carga, no tenga que compartir más tiempo de procesador que la resolución del tiempo real y no tenga que hacer swapping. Si se usa una versión de *linux RT*, teniendo como prioridad los tiempos del Scicos en sincronía con los de tiempo real, el sistema puede estar realizando una gran cantidad de aplicaciones tipo real y tipo servidor además de tipo cliente.

4.2 Control RST

Para el control de esta planta se usa un PI digital implementado en una estructura RST. El controlador RST es una estructura que por su flexibilidad permite implementar diferentes esquemas de control y en general filtros[4][6].

4.3 Descripción de la planta a controlar

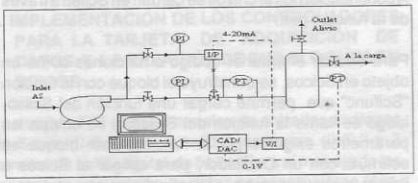


Figura 2. *Diagrama Esquemático - Planta de Presión PCT14*

El control se hace sobre la Planta de Presión Feedback PCT14, Figura 2. La materia prima de esta planta, el aire, la provee un compresor conectado directamente a ella. La planta posee un tanque acumulador, lugar donde se controla la presión a través de una servoválvula neumática. La válvula (que es el actuador) tiene un adecuador de señal de corriente a presión. La unidad está diseñada para operar con el PCT10, consola eléctrica que proporciona toda las alimentaciones de electricidad y adecuaciones de la señal.

4.4. Montaje modelador en Scicos

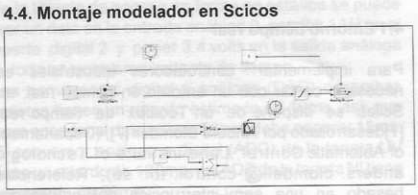


Figura 3.*Diagrama para modelar la planta*

La planta se modela a través de una técnica gráfica (no paramétrica), y a partir de ella se obtiene el modelo matemático (paramétrico), por el método de reacción usando para ello el Scicos cuya implementación se ilustra en la Figura 3. Si se observa el camino de lazo abierto se encuentra que simplemente se grafica la señal que se obtiene de la lectura de la planta, es decir, esta haciendo el trabajo de un osciloscopio en tiempo real y con la posibilidad de guardar esta información.
A partir de la curva de respuesta, se obtiene la función de transferencia de la planta:

$$Gp = e^{-0.5s}/(12s+1)$$

Se ve que el tiempo muerto para esta planta es 0.5s y el tiempo de estabilización es 12s apróximadamente. El ruido en la señal es muy pequeño y es producido por un error de filtrado en la salida de la tarjeta de adquisición de datos.

4.5 Implementación del controlador RST

A partir del modelo obtenido se procede a diseñar el controlador RST y se implementa con la herramienta gráfica Scicos de Scilab, el tiempo de muestreo seleccionado para el diseño es de 1.5 segundos, ver Figura 4.
Los datos de entrada son suministrados por el bloque "analog input" de la tarjeta de adquisición de datos al controlador y se obtienen algunos datos a la salida de la planta; aquí la señal es adecuada para poder hacer el control, se realimenta la bucla y en el lazo de realimentación se tiene el parámetro R. Lo siguiente que se observa después del sumador es la función del parámetro S que se encuentra en la bucla de lazo directo, esta señal sale a la planta mediante el bloque "analog output" de la tarjeta de adquisición al actuador; por otro lado el parámetro T se encuentra justo después de entrada de referencia, como es un prefiltro se encuentra antes del comparador.

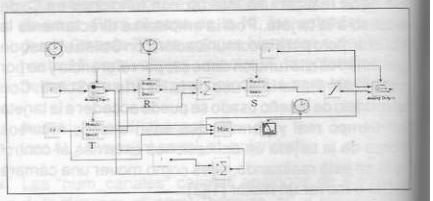


Figura 4. *Diagrama del sistema realimentado con controlador RST implementado en Scicos*

Las señales del graficador se muestran en la Figura 5 y corresponden a la señal del controlador (onda)

la señal de salida (señal mas pequeña en alterna) y la entrada de referencia (señal continua, en 0.66).
Para tiempos de muestreo muy pequeños el sistema se vuelve inestable; además, el controlador RST está diseñado para un tiempo de muestreo dado y esta implementado en este caso con una rata mucho menor.
La señal del controlador está saturada, la planta está protegida por medio del bloque que acota la señal entre cero y uno[5].
Como se puede ver el sistema es estable y la señal de control tiene un esfuerzo grande; la señal tiene un sobrepaso máximo del 30%, y el tiempo de estabilización es de aproximadamente 20 segundos.

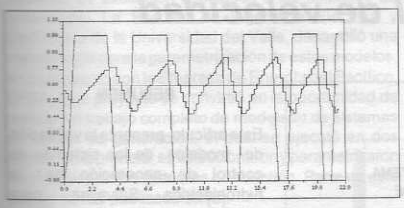


Figura 5. *Señal de salida 1. Tiempo de muestreo de 0.01 segundos*

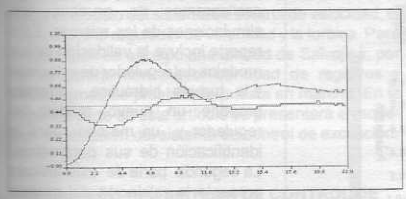


Figura 6. *Salidas con tiempo de muestreo de 1 segundo*

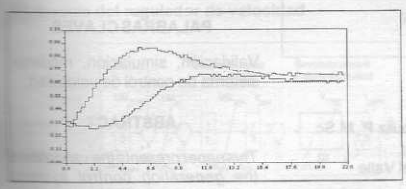


Figura 7. *Salidas con tiempo de muestreo 3 segundos*

El sistema tiene un sobrepaso máximo un poco mayor, aunque oscila menos y la señal de control es menos exigente; el tiempo de estabilización es de aproximadamente 17 segundos.

5. CONCLUSIONES

Hemos mostrado como a través de una herramienta de software GNU como Scicos de Scilab se puede obtener un controlador, graficar y obtener datos numéricos de señales y hacer el análisis de un sistema.
Se permite el trabajo de respuesta en tiempo real, lo cual no es posible en otras herramientas y fundamental en algunos procesos de control.
La forma de implementación de esta herramienta proporciona un acceso compartido a hardware con multiplexación de recursos y mejor aprovechamiento de los mismos.
Scilab y Scicos evolucionan desde una herramienta de simulación hacia un sistema de desarrollo e implementación de controladores industriales con muchas posibilidades en el ámbito de la academia, de la investigación e incluso para soluciones industriales.

6. BIBLIOGRAFÍA

[1] GOMEZ. Claude (Coordinating editor), Engineering And Cientific Computing Whit Scilab. Birkhäuser, 1999.
[2] Grupo Scilab INRIA. Manuales On-ine.
[3] Kamal B. Rojiani, Programming In C With Numerical Methods for Engineers. Prentice Hall, 1996
[4] Karl J. Astrom, Sistemas Controlados por Computador, Paraninfo, 1988.
[5] Joan Doré Landau, Identification et Comande des Systemes, Hermes, 1993
[6] Benjamin C. Kuo, Sistemas de Control Digital, CECSA, 1997.
[7] Anders Blumdell, Real Time Toolbox for Scilab.

AUTORES

Edinson Franco Mejía. Ingeniero Electricista (1992) y Magíster en Automática (1994) de la Universidad del Valle. Profesor Titular de la Escuela de Ingeniería Eléctrica y Electrónica de la Universidad del Valle. Actualmente adelanta estudios doctorales en Control de velocidad del motor de inducción sin sensor de de velocidad ni de posición.
efm@eiee.univalle.edu.co
efranco@control-automático.net

Carlos Andrés Ramos Paja. Ing. Electrónico de la Universidad del Valle (2002), estudiante de la Maestría en Automática de la Universidad del Valle
.

Angela María Maya,
Ingeniera Electrónico de la Universidad del Valle (2003).