Plataforma de Simulación de un Sistema Real de Control de Niveles Basada en Software Libre

Loyarte, Ariel Sebastián

Laboratorio de Sistemas de Control (LSC)
Universidad Tecnológica Nacional, Facultad Regional Santa Fe

Abstract
Se desarrollan las etapas implementadas en la construcción de un Laboratorio Virtual (LV) como aplicativo simulador de un sistema real de control de niveles, cuya premisa fundamental es la utilización íntegra de software libre, gratuito y multiplataforma. La planta está constituida por dos tanques de agua interconectados, más un tercer tanque reservorio. El modelo matemático correspondiente fue obtenido en un trabajo previo, en consideración de las estrategias de estimación de los parámetros inicialmente desconocidos. El LV se desarrolla en Python, utilizando los paquetes de módulos matemáticos NumPy y SciPy para la resolución numérica del sistema de ecuaciones diferenciales resultante. La interfaz gráfica se construye utilizando el toolkit GTK+ y el binding Python correspondiente. El paquete Matplotlib se utiliza para el trazado de los gráficos.

Palabras Clave
Laboratorio Virtual - Educación en Control - Modelado Matemático - Software Libre

Introducción

La experimentación en la educación en ingeniería resulta relevante en el transcurso de la carrera. Complementa a los desarrollos teóricos y acerca al estudiante a la implementación real de los conocimientos adquiridos durante el cursado. Permite además ejemplificar su aplicación en un futuro entorno laboral [1].

Experimentar implica la disponibilidad de determinados recursos, tales como el espacio físico para el laboratorio, los equipos a utilizar, recursos humanos, y el tiempo necesario para ejecutar tales prácticas, en consideración de un número reducido óptimo de estudiantes que conformen el grupo o comisión de trabajo. El conjunto de estos recursos y las prácticas allí desarrolladas caracterizan a un *Laboratorio Real* (LR), y se corresponde con la percepción habitual del concepto de *laboratorio*.

No obstante, el paradigma de la educación actual contempla la posibilidad de complementación de tales prácticas con la ejercitación sobre una plataforma de software que simule el desempeño de cada uno de los componentes de un LR [2]. En este caso, los recursos requeridos cambian. El estudiante interactúa con el software con la finalidad de reproducir los ensayos realizados sobre el LR, o bien con objeto de ensayar casos alternativos a los realizados en clase.

La forma de implementación de un laboratorio simulado es concebida como *Laboratorio Virtual* (LV) [3]. Un LV puede ser accedido por el estudiante en la comodidad de su hogar y en la cantidad de veces que éste crea necesaria o conveniente. Los recursos a utilizar para la conformación de un LV no sólo responden a las herramientas de software necesarias; sino que exigen el planteo y resolución de un problema matemático, dado por la necesidad de modelar el sistema que pretende ser simulado [4].

El modelado matemático del sistema es en sí mismo un ejercicio. Implica el estudio del sistema y de las leyes físicas involucradas, como así también la proposición de metodologías de ensayos que permitan estimar los parámetros desconocidos. Esto último frecuentemente resulta en un problema de optimización, y de aquí el inminente análisis de las variantes en cuanto a algoritmos capaces de su resolución.

Por otra parte, la naturaleza de los fenómenos involucrados en el sistema definirán las áreas temáticas del conocimiento. Resulta entonces que el ejercicio de modelado puede tratarse como interdisciplinario, cuando sean requeridos conocimientos de diversas áreas [5].

De esta manera, un LV constituye una valiosa herramienta para el estudiante, complementando la práctica experimental sobre el LR [6]. Pero al mismo tiempo, su desarrollo implica la resolución de un problema matemático complejo y su combinación con un ejercicio de programación.

Desde el punto de vista de la programación, el lenguaje utilizado debe ser apto para un tratamiento matemático muchas veces exigente. Por este motivo que aplicaciones como Matlab suelen destacarse por sobre otras [7]. Matlab incluye un lenguaje asociado de tipo scripting, de sencilla sintaxis y con una amplia colección de funciones matemáticas, dado su enfoque específico al cálculo numérico. No obstante, el cargo comercial requerido para el uso del programa puede resultar prohibitivo. Por lo que surge, ante estos casos, la necesidad de alternativas con características similares. Entre ellas, Octave y Scilab son las más frecuentemente utilizadas. Se trata de otras aplicaciones de cálculo numérico, pero libres y gratuitas.

Sin embargo, el usuario final, y sólo por una cuestión de usabilidad, deberá acceder al LV mediante una interfaz gráfica. Y ambos programas citados carecen de una herramienta que permita combinar su poder de cálculo con las capacidades gráficas necesarias.

Una posible solución es la utilización de un lenguaje de programación de uso general que pueda combinarse con complementos gratuitos que incluyan las funciones matemáticas precisadas. Y en este contexto Python demuestra ser una excelente solución [8,9]. Se trata de un lenguaje similar en muchos aspectos al utilizado por Matlab y, por ende, facilita la transición para quienes están habituados a este último.

Existe una amplia variedad de módulos adicionales orientados a su uso en el campo científico [10], que en su mayoría resultan gratuitos y libres; entre ellos se destacan NumPy y SciPy [11].

Por otra parte, resulta sencillo a través de los bindings correspondientes, implementar la interfaz de usuario por medio de alguno de los toolkits gráficos gratuitos disponibles, como GTK+ o Qt, entre otros.

En el presente trabajo se describen las etapas del desarrollo de un LV que simula el desempeño de un sistema real de control de niveles, instalado en el Laboratorio de Sistemas de Control (LSC) de la Facultad. El mismo está compuesto por una planta de dos tanques de agua interconectados, más un tercer tanque que acumula el líquido, que circula en un circuito cerrado. El sistema es complementado con los equipos y el instrumental asociado para la conformación de un lazo realimentado de control, con objeto de controlar el nivel de uno de los tanques, bajo la aplicación de un controlador PID.

Se resume el planteo del modelo matemático del sistema y, a continuación, las estrategias utilizadas en la estimación de los parámetros desconocidos (motivo de trabajos previos) [12,13].

Finalmente, se describe su implementación en Python con uso exclusivo de software libre. Se emplea el módulo de cálculo NumPy, y se utilizan los métodos de resolución de sistemas de ecuaciones diferenciales provistos por SciPy. Por otra parte, Matplotlib se emplea como módulo de trazado de gráficos en Python [14], mientras que GTK+ se utiliza para la interfaz gráfica, en conjunto con el binding PyGTK (o Python-GTK).

El aplicativo desarrollado es compatible con sistemas MS Windows y GNU/Linux, dada la característica de multiplataforma de cada componente de software utilizado.

Descripción de la planta

La planta está conformada por dos tanques de agua, TK_1 y TK_2, vinculados mediante un conducto que los interconecta desde sus

bases (Fig. 1). Cada uno de ellos posee a su vez un conducto individual de descarga sobre el tanque TK_3, que actúa como reservorio o acumulador. Desde este último, una bomba centrífuga toma el líquido impulsando el caudal de alimentación del sistema, hacia alguno de los otros tanques a elección.

La bomba es accionada por un motor eléctrico trifásico, cuya alimentación en tensión alterna deviene de un equipo variador de frecuencia. Éste recibe la tensión de la red, y mediante un circuito electrónico específico, genera una nueva terna de tensiones trifásicas, pero con una frecuencia que puede ser especificada en las configuraciones del mismo equipo. La frecuencia resulta ser aproximadamente proporcional a la velocidad de giro del motor, regulando por ende, el caudal impulsado por la bomba.

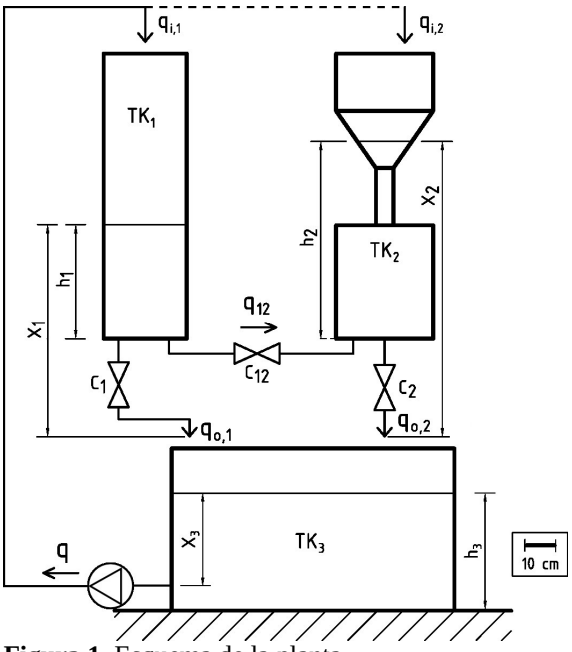

Figura 1. Esquema de la planta.

Modelo matemático de la planta

Se plantea para la planta un modelo matemático de primeros principios. Por cada tanque se plantea una ecuación de balance volumétrico, en la cual los caudales de descarga son relacionados con los niveles de líquido, a partir de la aplicación de las ecuaciones de Bernoulli y de Continuidad, características de la dinámica de fluidos.

El análisis riguroso de cada etapa de dicho planteo es realizado en un trabajo previo [12]. Se transcribe aquí el sistema de ecuaciones diferenciales obtenido.

$$A_1 \cdot \frac{dX_1}{dt} = q_{i,1} - C_1 \cdot \sqrt{X_1} + \qquad (1)$$
$$-C_{12} . Sgn\left(X_1 - X_2 + 3{,}3\ [cm]\right) . \sqrt{\left|X_1 - X_2 + 3{,}3\ [cm]\right|}$$

$$A_{2i} \cdot \frac{dX_2}{dt} = q_{i,2} - C_2 \cdot \sqrt{X_2} + \qquad (2)$$
$$+C_{12} . Sgn\left(X_1 - X_2 + 3{,}3\ [cm]\right) . \sqrt{\left|X_1 - X_2 + 3{,}3\ [cm]\right|}$$

Los niveles de líquido, considerados en este caso como variables de salida del sistema, son medidos con respecto a las correspondientes bases de cada tanque (h_1 y h_2). Sin embargo, el desarrollo de Bernoulli señala que los niveles deben ser medidos respecto de los puntos de descarga a presión atmosférica (X_1 y X_2). No obstante, pueden considerarse las siguientes equivalencias:

$$h_1 = X_1 - 18{,}0[cm] \qquad (3)$$

$$h_2 = X_2 - 21{,}3[cm] \qquad (4)$$

Las variables $q_{i,1}$ y $q_{i,2}$ corresponden a los caudales de alimentación de los tanques TK_1 y TK_2, respectivamente. Por otro lado, A_1 y A_{2i} representan el área de la sección transversal de cada uno de ellos, considerando que TK_2 presenta una sección no-uniforme. Estos últimos se corresponden con los parámetros inicialmente conocidos; mientras que C_1, C_2 y C_{12}, inicialmente desconocidos, representan las restricciones a la circulación del líquido en cada uno de los conductos de descarga, y deberán ser estimados posteriormente.

El sistema de alimentación, constituido por la bomba centrífuga, el variador de frecuencia y el tanque TK_3, es modelado separadamente. Este modelo permite definir el caudal inyectado a la planta.

Experiencias realizadas en el laboratorio demostraron que dicho caudal no sólo está influenciado por la frecuencia establecida por el equipo variador, sino también por la

presión de admisión de la bomba (proporcional al nivel X_3). Este modelo es desarrollado también en un trabajo previo [4], donde se detalla el procedimiento experimental utilizado.
Para los fines del LV, debe considerarse que el sistema de alimentación puede ser implementado como una tabla de doble entrada $q_i = f\ (Frecuencia,\ X_3)$, mediante cálculos de interpolación.

Estimación de parámetros desconocidos
El modelo de la planta quedará perfectamente definido cuando los parámetros C_1, C_2 y C_{12} sean valorados (estimados).
En un trabajo anterior [12], se analizaron dos variantes como metodologías de estimación. La primera de ellas consistió en el diseño de una serie de ensayos particulares sobre la planta, con una combinación específica de cierre y apertura de válvulas, que permitió aplicar finalmente un ajuste por el método de mínimos cuadrados.
En segunda instancia, se propuso la aplicación de un algoritmo de optimización por enjambre de partículas (PSO) [13]. Para este caso, el modelo fue implementado previamente en Matlab/Simulink, de modo que el algoritmo operaba con objeto de minimizar las diferencias entre las curvas reproducidas por el modelo y una serie de curvas medidas experimentalmente.

Lazo cerrado de control
El LV deberá reproducir el comportamiento del sistema, operando en un esquema de control realimentado, como lo hace el LR.
En el LR los niveles de cada tanque son medidos mediante sensores de nivel por presión. Las medidas resultan proporcionales a las corrientes eléctricas emitidas por cada sensor, y son recibidas por sendas entradas analógicas de un Controlador Lógico Programable (PLC), según se observa en la Fig. 2.
El PLC ha sido programado para ejecutar un algoritmo de control PID, en base a un valor de consigna (nivel deseado para alguno de los tanques), incluido como variable interna del mismo programa. De esta manera, la salida del controlador, coincidente con una salida analógica del PLC, actúa a su vez como entrada analógica del variador de frecuencia (operando aquí como *actuador*). Este último define la frecuencia de alimentación de la bomba a partir de una relación lineal entre la misma y dicha señal recibida.
La bomba centrífuga constituye el *órgano de acción final* del sistema, definiendo el caudal de alimentación (entrada para la planta) a partir de la velocidad de giro de su motor (función de la frecuencia) y del nivel que en todo momento posea el TK_3.
El PLC se encuentra comunicado con una PC del laboratorio mediante protocolo MODBUS sobre conexionado Ethernet, siendo posible modificar en cualquier momento el valor de consigna o la elección del tanque cuyo nivel se controla, como así también los parámetros que sintonizan al controlador.

Elección de las herramientas de software
La elección de los componentes de software para el desarrollo del LV está restringida al objetivo propuesto de utilización íntegra de software libre y gratuito.
Deben tenerse en consideración dos aspectos bien diferenciados: la resolución numérica del sistema de ecuaciones diferenciales resultante; y la implementación de la interfaz gráfica de usuario.

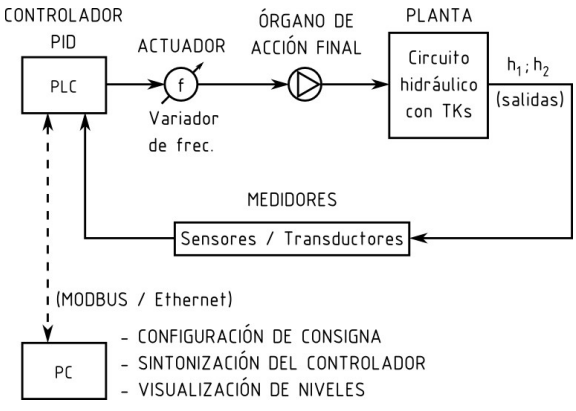

Figura 2. Esquema del lazo de control.

Para el primer punto, se ha probado con éxito el desempeño de la función *odeint*, incluida en el módulo *integrate* de integración numérica, perteneciente al paquete SciPy. Dicha función resulta en una implementación en Python del método *LSODA*, originalmente incluido en el paquete *ODEPACK* para Fortran.
Se trata de un método de paso variable, capaz de resolver sistemas de ecuaciones diferenciales ordinarias (EDOs), que incluye un algoritmo de detección de sistemas de EDOs rígidos ('*stiff*'). Inicialmente, aplica un método estándar ('*non-stiff*'); pero si el cálculo requiriera modificar los pasos de integración entre valores notoriamente diferentes (para evitar inestabilidades numéricas y optimizar tiempos), cambia automáticamente a un método del tipo '*stiff*'.
SciPy requiere de forma excluyente del paquete NumPy, que provee una amplia colección de funciones matemáticas al intérprete Python. Además, NumPy brindará las herramientas necesarias para el planteo del problema y cualquier cálculo de preparación de los datos o post-tratamiento de las salidas (soluciones de las EDOs).
Por otra parte, Matplotlib es un completo paquete de módulos orientados al trazado de todo tipo de gráficos en Python. Disponiendo de las dependencias adecuadas, Matplotlib puede mostrar cualquier gráfico en una ventana pre-diseñada, que además del gráfico, incluye una barra de herramientas con las opciones típicas de zoom, desplazamiento y exportación en diversos formatos de imagen. Incluye además el módulo *PyLab*, que permite controlar dicha ventana de gráficos con una sintaxis muy similar a la de Matlab.
Matplotlib permite también embeber sus gráficos y barra de herramientas en ventanas diseñadas separadamente por el programador. En estos casos, la sintaxis utiliza la orientación a objetos de Python.
En cuanto a la interfaz de usuario (IU), entre las alternativas libres disponibles, se optó por la utilización de GTK+.
En principio, las librerías GTK+ son utilizadas por programas desarrollados en *C*; o en su defecto en *Vala*, un lenguaje específicamente creado para la utilización de GTK+.
No obstante, el binding PyGTK permite la utilización de tales librerías gráficas en Python. Brinda soporte completo a todas las posibilidades que ofrece GTK+.
Por otro lado, la estructura de la interfaz gráfica puede ser diseñada utilizando *Glade*. Se trata de una aplicación del mismo grupo de desarrolladores de GTK+ que permite construir la IU sin necesidad de la programación mediante código.
Glade genera un archivo XML que puede, mediante el uso de PyGTK, ser importado desde el fichero que controla la interfaz, de forma análoga a la de un módulo Python tradicional. De esta manera, el uso de código Python para el diseño de la interfaz queda relegado a circunstancias de mayor complejidad, como las de incorporación de los widgets de Matplotlib.

Resolución del sistema de EDOs

Resolver el sistema de EDOs planteado para la planta, implica simular su comportamiento. De aquí que su resolución numérica constituya el núcleo del LV.
La resolución de las EDOs es llevada a cabo por el método LSODA, a partir de la función *odeint* descrita en el punto anterior.
Dado que la planta constituye sólo un bloque del sistema (lazo completo de control), tras cada iteración del método debe calcularse una nueva salida para cada uno de los otros bloques. Esto implica que en cada iteración se volverá a calcular un nuevo valor para el caudal de alimentación, alterando la expresión de las EDOs.
La complejidad de la aplicación la función *odeint* radica en que, a primera vista, sólo es aplicable a casos de EDOs con entradas constantes, y con parámetros invariantes. Esto implicaría la imposibilidad de simulación de un lazo de control, puesto a que la entrada de la planta (caudal de alimentación) debe poder variar en el tiempo, de acuerdo a los efectos del

controlador. Además, no sería posible considerar valores de consignas que varíen a lo largo de la simulación, ni la variabilidad del área de la sección transversal de TK_2 (A_{2i}).
Se ejemplifica a continuación la forma de uso de *odeint* para casos de EDOs invariantes y con entradas constantes.

```
from scipy.integrate import odeint
from numpy import sqrt
A = 1.
C = 10.
q = 120.
X0 = 10.
Tiempo = arange(0.,101.,1.)

def f(X,t):
      Derivada= (1/A)*(q-C*sqrt(X))
      return Derivada
X_salida = odeint(f,X0,Tiempo)
```

Por simplificación se utilizó la EDO que caracteriza solo a un tanque:

$$A \cdot \frac{dX}{dt} = q - C \cdot \sqrt{X} \qquad (5)$$

donde q es el caudal de entrada, C es el parámetro de restricción a la descarga, y X el nivel medido desde el punto de descarga.
En las primeras líneas se especifican los parámetros del sistema (invariantes), la variable de entrada (*q*, constante) y la condición de nivel inicial *X0*.
La función *f* corresponde a la forma más conveniente de definir la EDO. Ésta será accedida por *odeint* cada vez que sea requerida (última línea), y determinará el valor de la derivada de la variable incógnita en el tiempo *t*, y para un nivel *X*. Los valores de *t* y *X* serán proporcionados por el método LSODA de *odeint*.
Tiempo corresponde a una lista Python que indica el período de simulación y el paso.
Debe tenerse en cuenta que *Tiempo* no sólo define el intervalo de tiempo a simular, sino que además define un paso de integración propuesto (y estimativo) al método LSODA. Éste aplicará un algoritmo de paso variable, en el que el paso propuesto ayudará a determinar el grado de exactitud pretendido. Así, un paso sugerido bajo implicará pasos variables reales aplicados también bajos, y viceversa para pasos grandes.
Finalmente, *odeint* devuelve un vector *X_salida* que resulta de interpolar linealmente los valores de la variable de salida obtenidos a paso variable con los valores de tiempo alistados en *Tiempo*.
Si el sistema de EDOs no es de primer orden (caso del LV propuesto), deberán definirse dentro de *f* tantas expresiones de derivadas como orden tenga sistema. Esto implica que un sistema de orden *n* debe ser descompuesto en *n* EDOs de primer orden.
De esta manera, la función *f* devolvería el resultado de las *n* derivadas (una por cada EDO de primer orden), a partir de una lista Python. Y consecuentemente, la variable *X0* deberá ser una lista de las *n* condiciones iniciales; mientras que *X_salida* alistará los *n* vectores de salida (variables de estado).
Ante EDOs variantes y entradas variables, puede procederse de la siguiente manera:

```
from scipy.integrate import odeint
from numpy import sqrt, interp
C = 10.
q1 = 60.
X0 = 10.
Tiempo = arange(0.,101.,1.)
T_LSODA = []
q_total = []

def f(X,t):
      if X>40.:
            A = 1.
      else:
            A = 1.5

      if t>=30.:
            q2 = 70.
      else:
            q2 = 10.
      Derivada= (q1+q2-C*sqrt(X))/A
      if T_LSODA==[] or t>=T_LSODA[-1]:
            T_LSODA.append(t)
            q_total.append(q1+q2)
      return Derivada
X_salida = odeint(f,X0,Tiempo)
q_total=interp(Tiempo,T_LSODA,q_total)
```

En este caso, el tanque se alimenta con un caudal *q1* constante y uno *q2* variable (en función del tiempo). El parámetro *C* es invariante, mientras que *A* depende del nivel *X* (caso de sección no uniforme).

Se pretende determinar el nivel del líquido en el tiempo (*X_salida*), como así también el caudal total inyectado (*q_total*).
Los parámetros y entradas constantes pueden definirse fuera de *f*, de modo de no re-calcularse en cada iteración. Los parámetros variantes, entradas variables y variables internas deben ser definidos dentro de *f* (se calculan en cada paso).
Si se pretende almacenar la variación en el tiempo de algún parámetro o variable definida dentro de *f* (caso de *q_total*), puede aplicarse el método señalado, en el que se inicializa una lista vacía asociada a dicha variable y otra que almacenará los valores de tiempo utilizados por LSODA (*T_LSODA*), necesaria dado que el método aplicado es de paso variable.
El último condicional actualiza cada una de las listas mencionadas, incorporando el nuevo valor. El condicional responde a que en ocasiones LSODA utiliza pasos negativos. De esta manera (método simplificado), sólo se almacenan los valores pretendidos cuando el tiempo para el cual se evalúan las derivadas es posterior al último registrado en *T_LSODA*.
La última línea interpola para que *q_total* presente las mismas abscisas que *X_salida*.

Diseño de la interfaz de usuario (IU)
La IU se diseñó mediante Glade, por medio de GTK+. Su funcionamiento es controlado por un script Python gracias a PyGTK.
La ventana principal (Fig. 3) incluye cinco (5) pestañas. La primera define el tiempo de simulación y paso propuesto, más las condiciones de nivel iniciales.
La segunda pestaña permite configurar la consigna (constante, escalón o doble escalón) e indicar el nivel a controlar.
La tercera accede a la sintonización del controlador (PID), pudiéndose seleccionar entre sus modos *posicional* y *de velocidad*.
La cuarta pestaña permite definir un caudal de perturbación, decidir el tanque sobre el cual inyectarlo y su variación en el tiempo (escalón o doble escalón).
En la última pestaña se accede a opciones extra, como ser los límites inferior y superior de salida del variador de frecuencia, y la posibilidad de simular ruido (aditivo) de las mediciones, modelados a partir de una distribución normal con media y varianza a especificar.
La ventana de gráficos (Fig. 4) muestra las variables de interés en pestañas, e incluye la barra de herramientas de Matplotlib.
Entre otras opciones, se encuentra la posibilidad de la carga automática de valores de ejemplo, exportación de configuraciones y de un archivo con los valores de todas las variables del sistema.
Es posible alterar los parámetros de descarga, simular el cierre de válvulas, y simular la evolución paulatina de todas las variables en el tiempo. En este modo, el usuario debe especificar la escala temporal. Así por ejemplo, una escala de 0,05 implica que el tiempo de simulación será del 5% del que tomaría el ensayo real; mientras que en una escala unitaria el usuario vería las curvas trazándose en tiempo real.

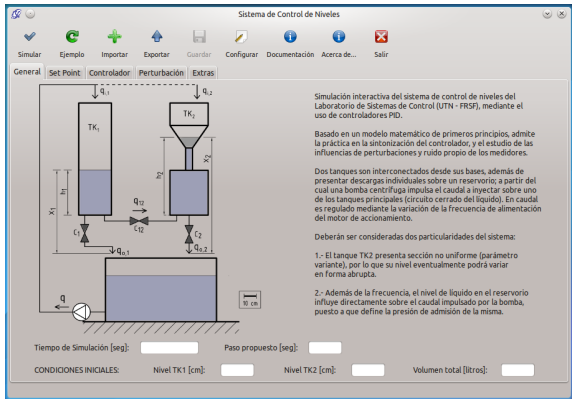

Figura 3. Ventana principal de la aplicación del LV.

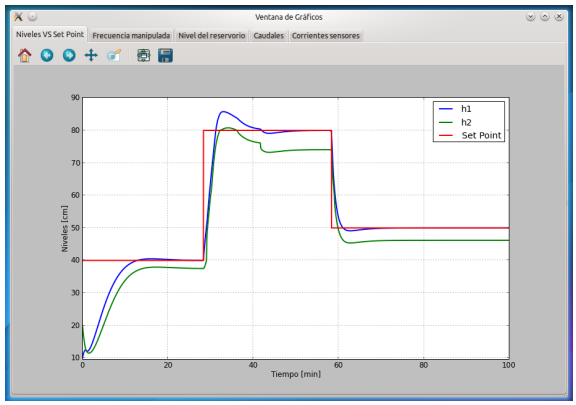

Figura 4. Ventana de gráficos del programa.

Conclusiones

El aplicativo desarrollado cumple satisfactoriamente con su cometido. El usuario (habitualmente alumno) podrá instalar el programa en su computadora personal, sin incumplir ningún tipo de licencia. Además, al ser el desarrollo considerado como software libre, será puesto a disposición su código fuente, factible de ser alterado, corregido, o servir de base o de ejemplo para otras aplicaciones similares.

En términos de su alcance, la aplicación es específica del área de control automático, y puntualmente simula el comportamiento de un sistema disponible en un laboratorio de la facultad. Sin embargo, la metodología seguida en su construcción y la evaluación de cada componente de software empleado es útil para múltiples aplicaciones que impliquen: resolución numérica de sistemas de ecuaciones diferenciales ordinarias (variantes y con entradas variables), y la conformación de una interfaz gráfica que evite el conocimiento en detalle, por parte del usuario, de la programación utilizada.

La metodología aquí descrita permite reemplazar en muchas ocasiones a programas privativos y comerciales como el caso de Matlab. Si bien este último permite una programación visual de rápido aprendizaje a partir de Simulink, el procedimiento aquí indicado para Python es suficientemente simple y ordenado como para reemplazar al anterior en la mayoría de los casos. Además, la modularidad de Python y la disponibilidad de una enorme variedad de complementos libres y gratuitos, hace posible extender sus posibilidades hacia límites muy amplios.

Referencias

[1] Feisel, L., Rosa, A. (2005). The Role of the Laboratory in Undergraduate Engineering Education. J. of Eng. Educ., 94, 121-130.

[2] Dormido, S. (2004). Control Learning: Present and Future. Annual Reviews in Control, 28 (1), 115- 136.

[3] Vary, J. (2000). "Informe de la Reunión de Expertos sobre Laboratorios Virtuales", Instituto Internacional de Física Teórica y Aplicada (IITAP), Ames, Iowa – UNESCO, París.

[4] Loyarte, A., Blas, M. (2012). Laboratorio Virtual Remoto: Una Herramienta para La Enseñanza de Control Automático. XXIII AADECA, 03-05 de oct. de 2012, Buenos Aires.

[5] Blas, M., Loyarte, A. (2012). Desarrollo e Implementación de un Laboratorio Virtual y Remoto: un Desafío Interdisciplinario. World Engineering Education Forum (WEEF) 2012. ISBN: 978-987-1896-03-5.

[6] Blas, M., Loyarte, A. (2012). Laboratorio Virtual y Remoto: Uso de la Tecnología de la Información Como Ayuda en la Educación. XIV Workshop de Investigadores en Ciencias de la Computación (WICC), Posadas, 26 y 27 de abril de 2012. ISBN E-Book: 978-950-766-082-5, 988-992.

[7] Blas, M., Loyarte, A. (2011). Integración de Java/Matlab para el desarrollo de un Laboratorio sin bienVirtual Remoto. Congreso Nacional de Estudiantes de Ingeniería en Sistemas de Información (CNEISI), Córdoba, 28 de septiembre al 01 de octubre de 2011.

[8] Oliphant, T. (2007). Python for Scientific Computing. Computing in Science & Engineering, 9 (3), 10-20.

[9] Ousterhout, J.K. (1998). Scripting: higher level programming for the 21st Century. IEEE Computer, 31 (3), 23-30.

[10] Millman, K., Aivazis, M. (2011). Python for Scientists and Engineers. Computing in Science & Engineering, 23 (2), 9-12.

[11] Jones, E., Oliphant, T., Peterson, P. y otros. (2001). SciPy: Open source scientific tools for Python. URL: http://www.scipy.org/.

[12] Loyarte, A., Diaz, G., Rosa, J. (2010). Sistema de Control de Niveles con Tanques de Sección No Uniforme. Modelado Matemático y Ajuste de Parámetros. XXII AADECA, 31 de agosto de 2010, Buenos Aires. Trabajo AF-1783.

[13] Loyarte, A. (2011). Sistema de Control de Niveles con Cuatro Tanques Interconectados: Modelado Matemático y Estimación de Parámetros. XIV Reunión del Procesamiento de la Información y Control (RPIC), Oro Verde, 16-18 de nov. de 2011, ISBN E-Book: 978-950-698-280-5, 965-970.

[14] Hunter, J. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9 (3), 90-95.

Datos de Contacto:

Ariel Loyarte. Facultad Regional Santa Fe (FRSF), Universidad Tecnológica Nacional (UTN). Lavaisse 610 (3000) Santa Fe (Argentina). E-mail: ariel.loyarte@gmail.com.

Trabajo de cátedra:

Cátedra de Control Automático, cuarto nivel de Ingeniería Eléctrica.