Universidad de Buenos Aires
Facultad De Ingeniería
Año 2019 - 2do Cuatrimestre

Laboratorio de Control Automático (86.22)

TRABAJO PRÁCTICO N^{o} 1
TEMA: S-Functions
FECHA: 11 de noviembre de 2019

INTEGRANTES:

Anastopoulos, Matias - #95120
<matias.anas@gmail.com>

Tomé, Juan Manuel - #94084
<tome.juanm@gmail.com>

Índice

1. Ejercicio 1: Modelo De Un Tanque

1.1. Ecuaciones matemáticas

Un tanque puede ser modelado mediante un único estado que represente su altura H. La ecuación de estado para el tanque está relacionada con el área del mismo A, y la diferencia entre el flujo de liquido que entra Q_i y que sale Q_o:

$$\dot{H} = \frac{1}{A}(Q_i - Q_o) \tag{1}$$

Por un lado, Q_i es una entrada del sistema, por otro lado, Q_o está relacionado con la diferencia entre la altura del tanque H y la altura del tanque siguiente en la cadena de tanques conectados H_{sig}. La relación entre los mismos esta dada por una resistencia de la conexión entre ambos tanques consecutivos (R):

$$Q_o = \frac{1}{R}\sqrt{|H - H_{sig}|}\ sign(H - H_{sig}) \tag{2}$$

De no tener un tanque consecutivo, se deberá tomar $H_{sig} = 0$.

Todos los tanques poseen una altura máxima H_{max}. Si se supera esa altura, el tanque se desborda y pierde líquido. Por lo tanto, hay ciertas condiciones en las que se debe tomar $\dot{H} = 0$:

1. $(Q_i > Q_o)$ y $H \geq H_{max}$
2. $(Q_i < Q_o)$ y $H \leq 0$

1.2. Implementación del modelo del Tanque

La implementación de la *s-function* de cada tanque se encuentra en el archivo **tanque.m**. El mismo tiene la siguiente interfaz:

```
function [sys,x0,str,ts,simStateCompliance] = tanque(t, x, u, flag, R, hmax, A, hini)
```

Donde los parámetros son:

- R: Resistencia de salida.
- hmax: Altura máxima.
- A: Area.
- hini: Altura inicial.

El esquema en *simulink* puede verse en la figura 1. El sistema posee dos entradas: La primera, es el caudal de entrada, la segunda, la altura del tanque siguiente o cero en caso de no utilizarse. Por otro lado, el sistema tiene dos salidas: La altura y el caudal de salida. Al momento de conectar varios tanques, se deberá conectar caudales de salida con caudales de entrada entre tanques consecutivos, y se deberá conectar la altura de salida del tanque siguiente a la entrada de altura del tanque anterior.

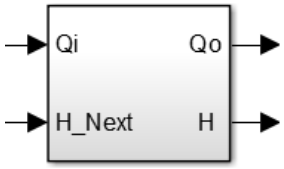

Figura 1: Esquema De Un Tanque.

1.2.1. Función De Estado

Todo lo anterior se encuentra codificado en la siguiente función de la *s-function*. La misma es la que se encarga de computar la derivada de los estados.

```
function sys=mdlDerivatives(t,x,u, R, hmax, A, hini)

%Estado.
h = x;

%Entradas.
qi = u(1);
h_next = u(2);

%Por un tema numerico, de ser necesario fijo las alturas en
    cero.
if(h <= 0)
    h = 0;
end

if(h_next <= 0)
    h_next = 0;
end

%Caudal de salida.
qo = 1 / R * sqrt(abs(h - h_next)) * sign(h - h_next);

%Computo Derivadas Del Estado.
delta_q = qi - qo;
```

```
if((delta_q > 0) && (h >= hmax))
    sys = 0;
elseif((delta_q < 0) && (h <= 0))
    sys = 0;
else
    sys = 1 / A * delta_q;
end
```

1.2.2. Función De Salida

La segunda función de importancia del bloque de la *s-function* es la que computa la ecuación de salida. A continuación se expone la función:

```
%Estado.
h = x;

%Entradas.
h_next = u(2);

%Por un tema numerico, de ser necesario fijo las alturas en
    cero.
if(h <= 0)
    h = 0;
end

if(h_next <= 0)
    h_next = 0;
end

%Computo Salidas.
qo = 1 / R * sqrt(abs(h - h_next)) * sign(h - h_next);

sys = [h, qo];
```

1.3. Casos De Prueba

A continuación se exponen varios casos de prueba que demuestran el funcionamiento de la *s-function*. En todos los mismos se ha se ha optado por utilizar el *solver* **ode23tb**. Se ha optado por utilizar este dado a que es de paso variable, condición necesaria para poder resolver el problema en un tiempo razonable con una resolución/cantidad de puntos aceptable. Además se ha optado por un *solver* de tipo *stiff*, que es mas apto para resolver problemas mal condicionados.

1.3.1. Caso 1: Tanque Con Caudal De Entrada Variable

El esquema utilizado en este caso es el de la figura 2.

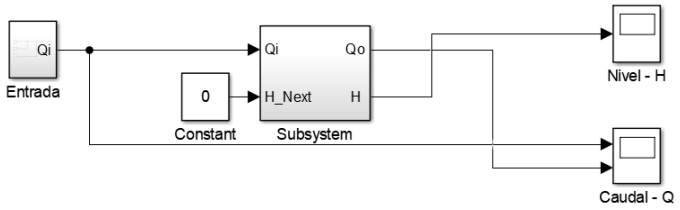

Figura 2: Tanque Único.

La secuencia del caudal de entrada es la siguiente:

1. Se incrementa de 0 a 1 en $t_0 = 1$s.
2. Se decrementa de 1 a 0 en $t_1 = 15$s.
3. Se incrementa de 0 a 0,5 en $t_2 = 25$s.

Se ha tomado como altura inicial nula y parámetros de áreas, resistencias y alturas máximas iguales a 1.

En la figura 3 se observa la evolución del nivel del tanque. Se observa que cuando el caudal de entrada está fijo en un valor distinto de 0, el nivel del tanque se incrementa hasta llegar a un equilibrio. Por otro lado, cuando el caudal de entrada está en 0, el nivel del tanque se decrementa hasta 0.

En la figura 4 se observan los caudales de entrada y salida del tanque. Se observa que, pasados los transitorios, ambos caudales siempre se igualan.

1.3.2. Caso 2: Tanque Que Se Desborda

El esquema utilizado en este caso es el de la figura 2.

La secuencia del caudal de entrada es la siguiente: La señal de entrada se incrementa desde un valor 0 a 1 en $t_0 = 1$, y se decrementa de 1 a 0 en un $t_1 = 15$.

Se ha tomado como altura inicial nula y parámetros de áreas, resistencias iguales a 1. Se ha puesto una altura máxima de 0,6 para que se desborde el tanque.

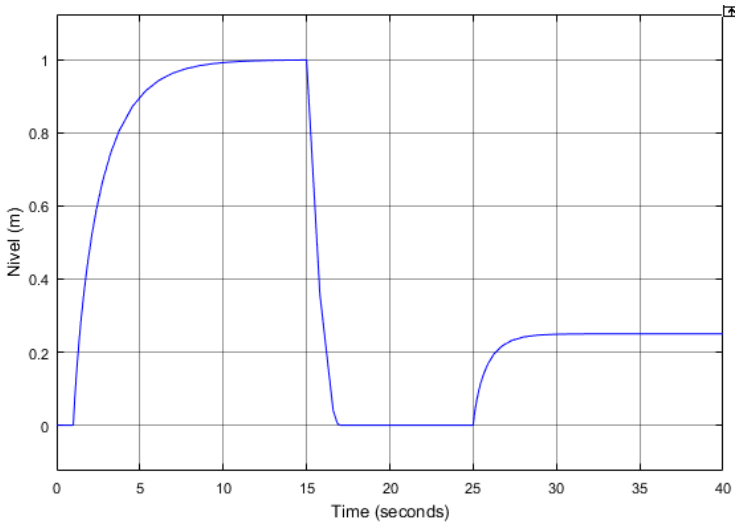

Figura 3: Caso 1: Tanque Único - Caudal De Entrada Variable - Nivel

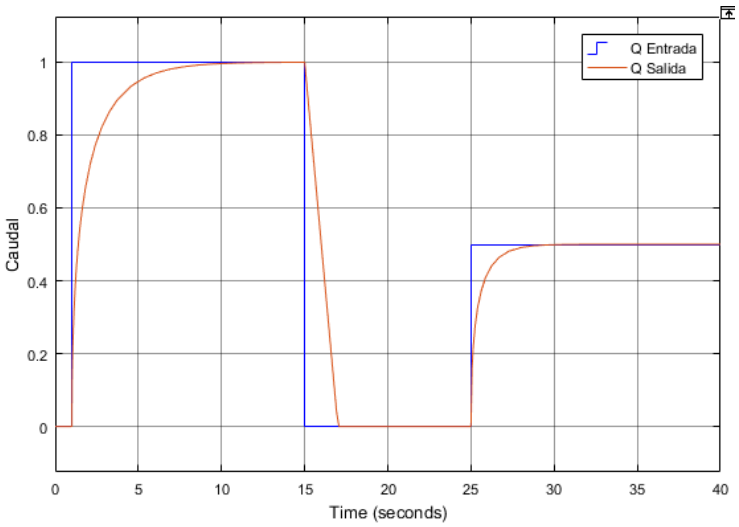

Figura 4: Caso 1: Tanque Único - Caudal De Entrada Variable - Caudales

En la figura 5 se observa la evolución del nivel del tanque. Se observa como en un determinado momento, la altura se satura en 0,6m.

En la figura 6 se observan los caudales de entrada y salida del tanque. Se observa que los caudales no se igualan dado a que hay una parte del caudal que se pierde en el desborde.

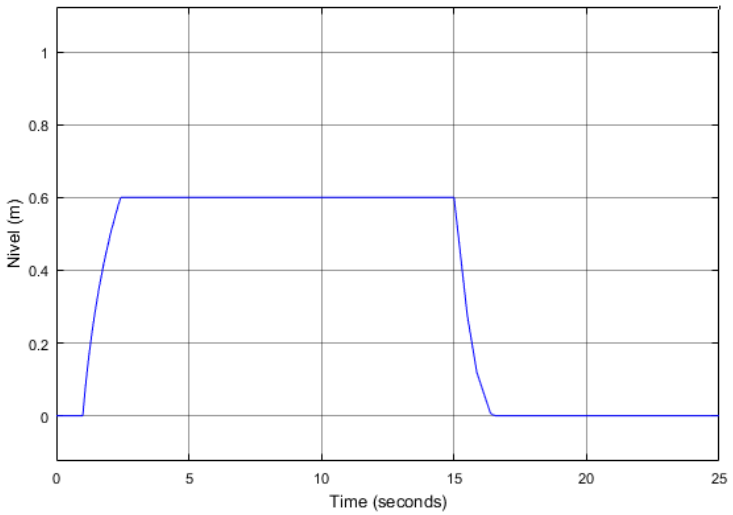

Figura 5: Caso 2: Tanque Único - Tanque Se Desborda - Nivel

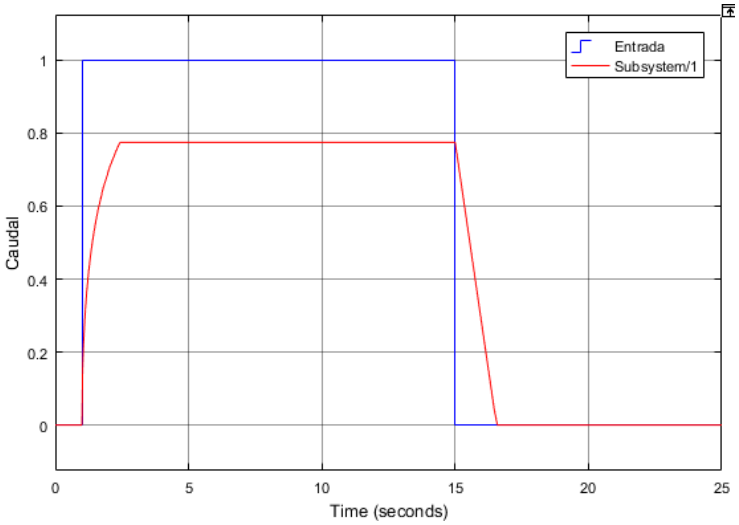

Figura 6: Caso 2: Tanque Único - Tanque Se Desborda - Caudales

1.3.3. Caso 3: Tanque Con Altura Inicial

El esquema utilizado en este caso es el de la figura 2. En este caso la entrada es nula en toda la simulación. Se ha tomado como áreas y resistencias iguales a 1, altura máxima de 1m y altura inicial igual a 1m.

En la figura 7 se observa la evolución del nivel del tanque. Se observa que arranca en 1m y el tanque se vacía hasta 0.

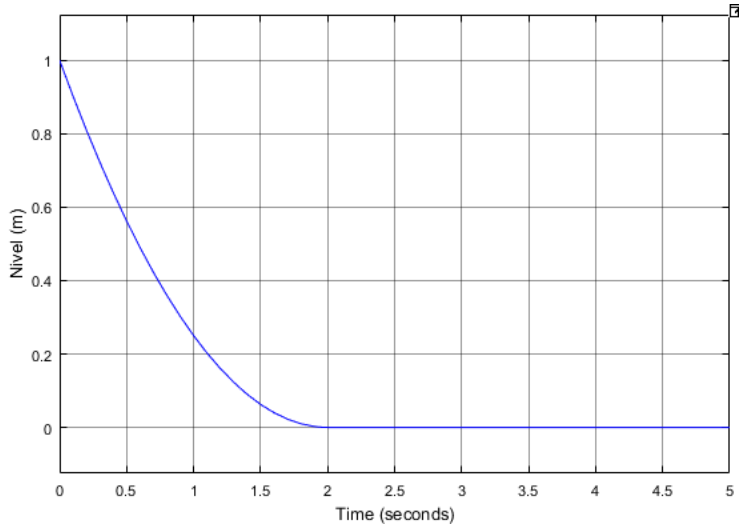

Figura 7: Caso 3: Tanque Único - Condición Inicial - Nivel

En la figura 8 se observan los caudales de entrada y salida del tanque. Se observa que el caudal de salida va disminuyendo hasta que el tanque se vacía.

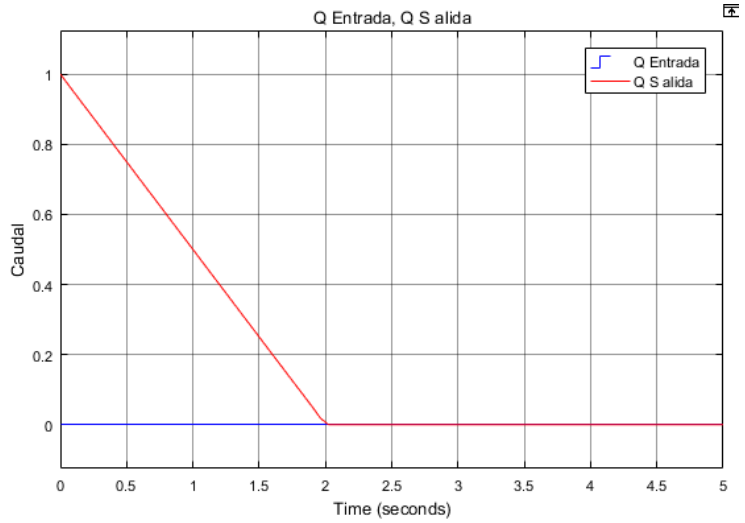

Figura 8: Caso 3: Tanque Único - Condición Inicial - Caudales

1.3.4. Caso 4: Tres Tanques En Cascada

El esquema utilizado en este caso es el de la figura 9. Esta es la forma de conectar tres tanques en cascada. Como se explicó, se conectan caudales de salida con caudales de entrada en tanques consecutivos. Por otro lado, se conecta la salida de nivel de cada tanque siguiente con la entrada de nivel de cada tanque anterior.

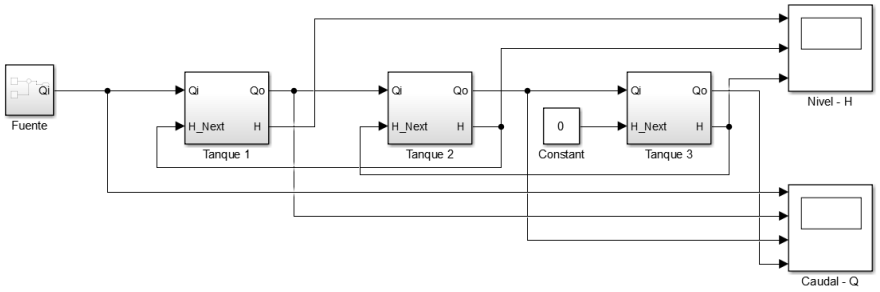

Figura 9: Caso 4: Tres Tanques En Cascada.

El caudal de entrada se incrementa de 0 a 0,2 en $t_0 = 1$, luego se lo decrementa a 0 en $t_1 = 20$. Se ha tomado condiciones iniciales de altura nulas para todos los tanques, para compararlos todos partiendo de un estado inicial común, y ver cuál se llena primero.

La evolución de los caudales puede observarse en la figura 10. Se observa como el primer tanque es el primero en llenarse y el ultimo en vaciarse.

Puede demostrarse que la altura final de cada tanque siguiente es inferior a la anterior. En equilibrio:

$$Q_i = Q_o \tag{3}$$

Por otro lado:

$$Q_i = Q_o = \frac{1}{R}\sqrt{H - H_{sig}} \tag{4}$$

Por lo tanto:

$$H_{sig} = H - (R\,Q_i)^2 \tag{5}$$

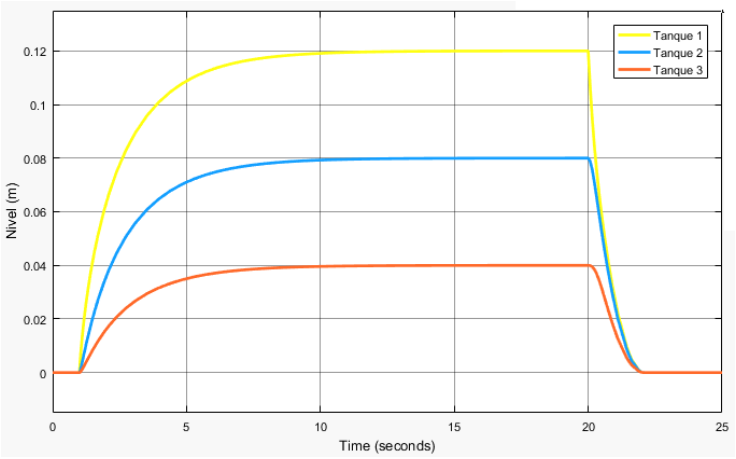

Figura 10: Caso 4: Tres Tanques En Cascada - Nivel

Y:

$$H_{sig} < H \tag{6}$$

2. Ejercicio 2: Identificación De Un Motor De Corriente Continua

Este punto tiene como objetivo poder realizar una identificación de una planta en particular. Se trata de un motor de corriente continua, cuya entrada es una tensión continua y cuyas salidas son pulsos de *encoder*, conectado al eje del motor.

La simulación de la planta está contenida en el archivo **dcmotor_qenc.p**, pero para poder realizar la identificación, será necesario programar otra **s-function** que convierta los pulsos del encoder en una medida de velocidad en revoluciones por minuto. Dicha función está contenida en el archivo **encoder_a_rpm.m**.

2.1. Encoder A RPM

La *s-function* que realiza la conversión de pulsos de encoder a RPM funciona de la siguiente manera: Cuenta la cantidad de flancos negativos o positivos de los pulsos de la primera de las señales de encoder, en una ventana de tiempo determinada.

El bloque posee dos tiempos relevantes: El primero, es cada cuanto muestrea la señal de encoder para poder detectar los flancos,este es el mayor de los dos, dado a que necesita muestrear a gran velocidad para poder detectar flancos. El segundo, es cada cuanto tiempo el bloque emite una medición y está relacionado con la ventana de tiempo en la cual el bloque lleva la cuenta de pulsos de encoder.

El bloque posee cuatro estados: El primero, se utiliza para recordar la cantidad de pulsos que lleva contados en la ventana. El segundo, se utiliza para retener la medición anterior mientras se esta contando los flancos en la nueva ventana. El tercero, es el que lleva el tiempo de la ventana transcurrido desde la medición anteriormente emitida. Y el cuarto guarda el estado anterior de la señal de encoder, para poder detectar cuando la misma cambia (flanco).

Una vez transcurrida la ventana, en el bloque de salida de la *s-function* emite la medición, que es la cantidad de pulsos multiplicada por un factor de escala para convertir a RPM.

A continuación se expone la función de estado de la *s-function* que convierte pulsos a RPM:

```
function sys=mdlUpdate(t,x,u, Ts, Tm, ppv_encoder)
    % Entradas del encoder.
    tension_A = u(1);
    tension_B = u(2);

    % Umbrales de deteccion.
    UMBRAL_TENSION_INFERIOR = 0.2;
    UMBRAL_TENSION_SUPERIOR = 0.8;
    ESTADO_BAJO = 0;
    ESTADO_ALTO = 1;


    % Estados.
    pulsos_acumulados = x(1);
    pulsos_anteriores = x(2);
    tiempo_acumulado = x(3);
    estado_A_anterior = x(4);

    if(estado_A_anterior == ESTADO_BAJO) && (tension_A >
   UMBRAL_TENSION_SUPERIOR)
        % Flanco positivo.
        pulsos_acumulados = pulsos_acumulados + 1;
        estado_A_anterior = ESTADO_ALTO;
    elseif(estado_A_anterior == ESTADO_ALTO) && (tension_A <
   UMBRAL_TENSION_INFERIOR)
        % Flanco negativo.
```

```
        pulsos_acumulados = pulsos_acumulados + 1;
        estado_A_anterior = ESTADO_BAJO;
    else
        % pulsos_acumulados no cambia.
        % estado_A_anterior no cambia.
    end

    % Ventana.
    tiempo_acumulado = tiempo_acumulado + Ts;
    if (tiempo_acumulado > Tm)
        tiempo_acumulado = 0;
        pulsos_actuales = pulsos_acumulados;
        pulsos_acumulados = 0;
    else
        pulsos_actuales = pulsos_anteriores;
    end

    sys = [pulsos_acumulados, pulsos_actuales,
   tiempo_acumulado, estado_A_anterior];

% end mdlUpdate
```

A continuación se expone la función de salida de la *s-function* que convierte pulsos a RPM:

```
function sys=mdlOutputs(t,x,u, Ts, Tm, ppv_encoder)
    pulsos_actuales = x(2);

    % Factor de escala.
    sys = pulsos_actuales / ppv_encoder * 60 / Tm / 2;
% end mdlOutputs
```

2.2. Modelo Propuesto

Se ha propuesto como modelo del sistema, un sistema de primer orden con un retardo temporal. Es decir:

$$T(s) = \frac{A}{s\tau + 1} e^{-t_r s} \tag{7}$$

La justificación del modelo se dará mas adelante basado en los resultados.

2.3. Ensayo

Para el ensayo de la identificación, se ha conectado en cascada el bloque del motor, junto con el bloque del conversor de encoder a RPM. Se ha utilizado como señal de entrada dos escalones diferidos en el tiempo. La amplitud de cada escalón es de 6V, dando un total de 12V, separados por tiempo de 0,25 segundos. El objetivo de poner dos escalones es poder realizar la identificación centrándonos en el segundo de los escalones. El objetivo del primero, es colocar al motor en una condición de velocidad inicial distinta de cero. El problema con las velocidades bajas es que se tienen pocos pulsos de encoder, y por lo tanto, una peor resolución.

El esquema de simulink utilizado está en el archivo **test_motor_2016a.slx**, y puede verse en la figura 11.

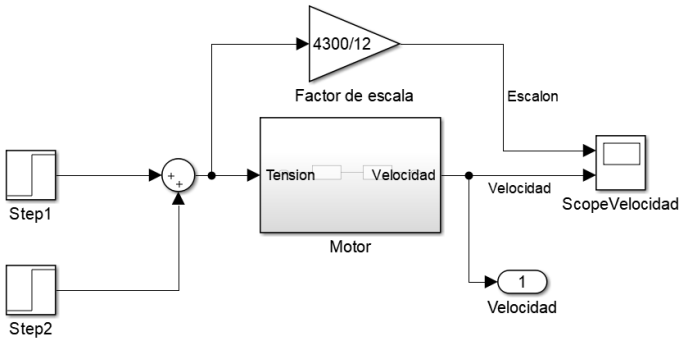

Figura 11: Ensayo Del Motor.

Como parámetros de la simulación se han utilizado los siguientes:

- Tiempo de muestreo de pulsos de encoder: Ts = 0,00001 segundos.
- Tiempo entre mediciónes: Tm = 0,005 segundos.
- Pulsos de encoder: ppv_encoder = 150.

La justificación de los pulsos de encoder y tiempo entre mediciones está basada en que la diferencia entre el modelo y la planta no supere el 5 % en error cuadrático medio. Un encoder de 150 pulsos por vuelta se encuentra en el mercado, aunque no son de los mas accesibles. Por otro lado, se optó por un tiempo de muestreo de pulsos que no pierda pulsos en el peor de los casos y que no genere una simulación que tarde mas de 10 segundos.

Se ha optado como elección del *solver* el **ode45** que, a pesar de ser una simulación de un sistema conmutado, dio resultados satisfactorios en cuanto a la forma de la señal y sus transiciones abruptas.

2.4. Resultados De La Simulación Y Estimación De Parámetros

El algoritmo que realiza la estimación de parámetros se encuentra en el archivo **ejercicio_2.m**. Para la estimación del τ se computó el tiempo que se tardaba en llegar al 63 % del valor final. El computo del valor de A es simplemente encontrar el factor de escala que hace que lleguen ambas curvas al mismo valor final. Finalmente no se detectó un t_r comparable con los tiempos entre los que se emiten mediciones, y por lo tanto se lo tomó como cero. Los resultados del algoritmo pueden verse en la tabla 1.

Parámetros	
τ	0,04s
t_r	0s
A	366,67 rpm/V

Tabla 1: Parámetros Estimados.

La comparación entre el modelo y la *s-function* puede verse en la figura 12. La raiz cuadráda del error cuadrático medio computado por el algoritmo es de 57,93 RPM, que representa un valor inferior al 5 %, comparándolo tanto con el valor inicial como final de las curvas.

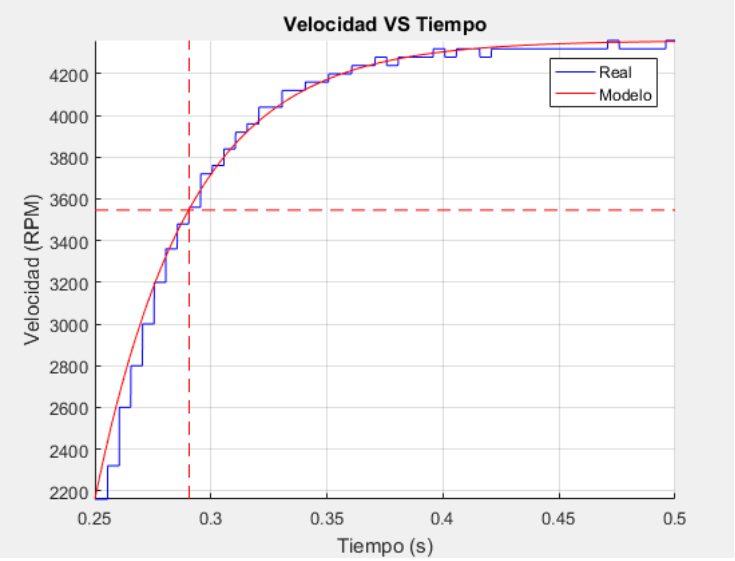

Figura 12: Comparación Modelo VS S-Function.

3. Ejercicio 3: Implementación De Un Controlador PID

El objetivo de esta sección es la de la implementación de un control a lazo cerrado para el motor caracterizado en el punto anterior. Para ello se procedió a la utilización de un controlador PID.
Para la realización del ensayo, se utilizó una *S-Function* como bloque controlador PID. La misma consiste de un bloque al cuál le entran simultáneamente la referencia deseada, $r(t)$ y la señal de salida $y(t)$ y cuya salida es la acción de control, *u(t)*. Un diagrama se presenta a continuación:

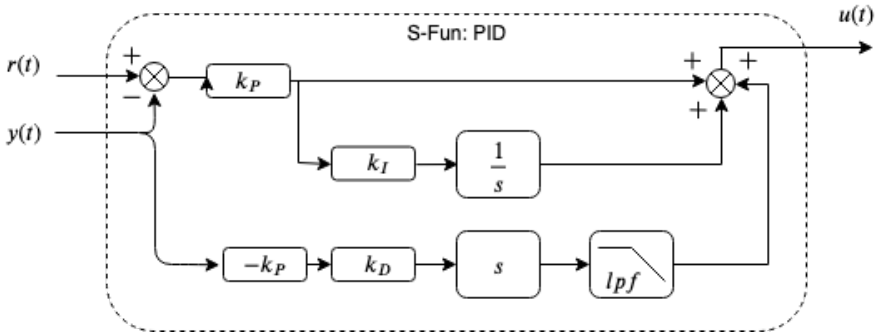

Figura 13: Bloque PID

Las ecuaciones que caracterizan a este PID son :

$$U(s) = k_P \left[R(s) - Y(s) + k_I \left(\frac{R(s) - Y(s)}{s} \right) + k_D \left(\frac{s}{1 + s\left(\frac{k_D}{N}\right)} \right) \right] \tag{8}$$

$$U(s) = k_P \left[R(s) - Y(s) \right] + k_P \; k_I \left(\frac{R(s) - Y(s)}{s} \right) + \frac{k_P}{k_D} \left(\frac{s \; Y(s)}{1 + s\left(\frac{k_D}{N}\right)} \right) \tag{9}$$

Donde: Parte proporcional:

$$P = k_P \left(R(s) - Y(s) \right) \xrightarrow{Discretizo} \boxed{P_{[k]} = k_P \left[r_{[k]} - y_{[k]} \right]} \tag{10}$$

Parte Integral:

$$I(s) \; s = k_P \; k_I \left[R(s) - Y(s) \right] \xrightarrow{tiempo} \frac{\partial I(t)}{\partial t} = k_P \; k_I \left[r(t) - y(t) \right] \rightarrow \tag{11}$$

Utilizando Forward Euler:

$$\rightarrow \frac{I_{[kh+h]} - I_{[k\ h]}}{h} = k_P\ k_I \left(r_{[k\ h]} - y_{[k\ h]}\right) \tag{12}$$

$$\rightarrow I_{[kh+h]} = I_{[kh]} + h\ k_P\ k_I \left(r_{[k\ h]} - y_{[k\ h]}\right) \tag{13}$$

$$\rightarrow \boxed{I_{[k+1]} = I_{[k]} + h\ k_P\ k_I \left[r_{[k]} - y_{[k]}\right]} \tag{14}$$

Parte Derivativa:

$$D(s)\left[1 + \frac{s\ k_P}{N}\right] = k_P\ k_D\ Y(s) \rightarrow D(s) + s\ D(s)\frac{k_D}{N} = k_P\ k_D\ Y(s) \tag{15}$$

$$\xrightarrow{tiempo} D(t) + \frac{\partial D(t)}{\partial t}\frac{k_D}{N} = k_P\ k_D\ \frac{\partial y(t)}{\partial t} \tag{16}$$

Utilizando Backward Euler:

$$D_{[kh]} + \left[\frac{D_{[kh+h]} - D_{[kh]}}{h}\right]\frac{k_D}{N} = k_P\ k_D\ \left[\frac{y_{[kh+h]} - y_{[kh]}}{h}\right] \rightarrow \tag{17}$$

$$\rightarrow D_{[k]}\left[1 + \frac{k_D}{Nh}\right] = \frac{D_{[k-1]}}{h}\ \frac{k_D}{N} + \frac{k_P\ k_D}{h}\left[y_{[k]} - y_{[k-1]}\right] \tag{18}$$

$$\rightarrow D_{[k]} = D_{[k-1]}\left[\frac{\frac{k_D}{Nh}}{1 + \frac{k_D}{Nh}}\right] + \frac{k_P\frac{k_D}{h}}{1 + \frac{k_D}{Nh}}\left[y_{[k]} - y_{[k-1]}\right] \tag{19}$$

Haciendo :

$$\gamma = \frac{k_D}{N} \tag{20}$$

$$\boxed{D_{[k]} = D_{[k-1]}\frac{\gamma}{\gamma + h} - \frac{k_P\ k_D}{\gamma + h}\left[y_{[k]} - y_{[k-1]}\right]} \tag{21}$$

De este modo al problema le quedan las siguientes entradas, salidas y estados:

- Entradas: $r_{[k]}$, $y_{[k]}$
- Salida : $u_{[k]}$
- Estados :
  - $I_{[k]}$,
  - $D_{[k-1]}$
  - $y_{[k-1]}$

Entonces el vector de estados es :

$$\bar{X} = \begin{bmatrix} I_{[k]} \\ D_{[k-1]} \\ y_{[k-1]} \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \tag{22}$$

3.1. Implementación

A continuación se explicarán las funciones de la *S-Function* relevantes al problema, tales como la actualización de estados y la actualización de la salida:

3.1.1. Función de Estado

```
function sys=mdlUpdate(t,x,u, kp, ki, kd, h, gamma)

    %Entradas r(k) e y(k)
    rk = u(1);
    yk = u(2);

    %estados I(k), D(k-1) y y(k-1)
    Ik   = x(1);
    Dkm1 = x(2);
    ykm1 = x(3);

    %hallamos el nuevo estado I(k+1)
    Ikp1 = Ik + kp * ki * h * (rk - yk);

    %hallamos el nuevo estado D(k)
    Dk = (gamma * Dkm1 - kp * kd * (yk - ykm1)) / (gamma + h);

    %actualizamos los estados que al estar en un nuevo momento
     pasan a ser
    % I(k+1) D(k) y(k)
    sys = [Ikp1, Dk, yk];
```

3.1.2. Función de salida

```
function sys=mdlOutputs(t,x,u, kp, ki, kd, h, gamma)
    %entradas r(k) e y(k)
    rk = u(1);
    yk = u(2);
```

```
%estados I(k) D(k-1) y(k-1)
Ik = x(1);
Dkm1 = x(2);
ykm1 = x(3);

%se obtienen los valores de salida
Pk = kp * (rk - yk);
Dk = (gamma * Dkm1 - kp * kd * (yk - ykm1)) / (gamma + h);

sys = Pk + Ik + Dk;
```

Esta *S-Function* posee como constantes de entrada el siguiente vector :

$$Ctes = \begin{bmatrix} k_P \\ k_I \\ k_D \\ h \\ \gamma \end{bmatrix} = \begin{bmatrix} \text{Constante proporcional} \\ \text{Constante integral} \\ \text{Constante derivativa} \\ \text{Paso numérico} \\ \text{frecuencia de corte del filtro del derivativo} \end{bmatrix} \tag{23}$$

Para obtener estas constantes se procede al cálculo del controlador PID por el método de *Ziegler-Nichols*.

3.2. Cálculo del controlador

Se ha optado como criterio de ajuste de los PID poder llegar al valor de referencia en el menor tiempo posible, limitando a que el sobrepico no exceda un 3 % y teniendo en cuenta que no se sature la tensión de entrada.
Los métodos empíricos de *Ziegler-Nichols* permiten obtener las constantes del controlador PID utilizando una tabla y la planta identificada.

Existen dos métodos, el primero es utilizado para plantas de primer orden. Y basan la relación de constantes para el cálculo del PID partiendo de que la planta posee un retardo temporal. Dado a que nosotros no hemos podido caracterizar dicho retardo fue necesario ir al seguundo método de *Ziegler-Nichols*.

El segundo método consiste en tomar la planta identificada a lazo cerrado y colocarle un controlador tipo proporcional, aumentando su valor hasta el punto en que el sistema empieza a oscilar y de este modo obtener el período de oscilación y con ambos datos de k_P critico y t_{osc} ir a una tabla provista por*Ziegler-Nichols*. Nuevamente nos encontramos con un problema, dado que nuestra planta identificada fue de primer orden no existió $k_P > 0$ que cumpliera con la pauta de oscilación, no obstante el segundo método de *Ziegler-Nichols* puede también ser utilizado de manera directa con la planta en cuestión, por lo tanto se procedió a realizar los mismos pasos recién mencionados

para la planta *real*, caracterizada por el *encoder* y la *S-Function* que caracterizaba los datos del mismo.

De este modo se obtuvieron un $k_{Pcr} = 0{,}036$ y un período de $T_{osc} = 20$ms. No obstante, cuando posteriormente se quiso obtener los valores correspondientes a las constantes del PID utilizando la tabla *Ziegler-Nichols*, estos valores no se encontraban ni de cerca de ser válidos y por ende fue necesario tocar los valores de k_P,k_I,k_D manualmente hasta que se logró las condiciones a lazo cerrado deseadas.

Diferencias entre la planta identificada y real: Se ha observado que la planta real posee una tendencia a oscilar mas que el modelo de primer orden propuesto. Por lo tanto, se requirió bajar la ganancia proporcional, con respecto a la ganancia utilizada con el modelo identificado, para poder lograr las especificaciones propuestas.

Cte	Planta Identificada	Planta Real
k_P	0.008	0.005
k_I	25	25
k_D	0.00125	0.00125
γ	1/10	1/10

En ambos casos se utilizo un tiempo de muestreo $T_s = h = 0{,}005$

3.3. Ensayos a lazo cerrado

A continuación se mostrará el diagrama de simulink para la planta a lazo cerrado, (figura 14)

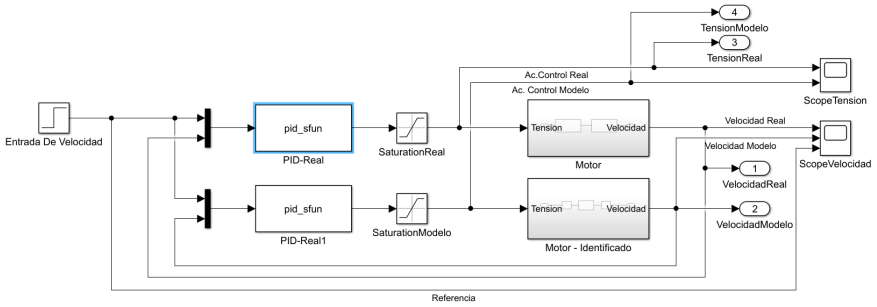

Figura 14: Diagrama de simulink, ejercicio 3

Aquí mostrarán las respuestas tanto de la salida del sistema, (figura 15) como de la acción de control (figura 16). Las curvas azules corresponden al modelo

identificado de la planta, y en rojo las que corresponden a la planta simulada con la *s-function* provista por la cátedra.

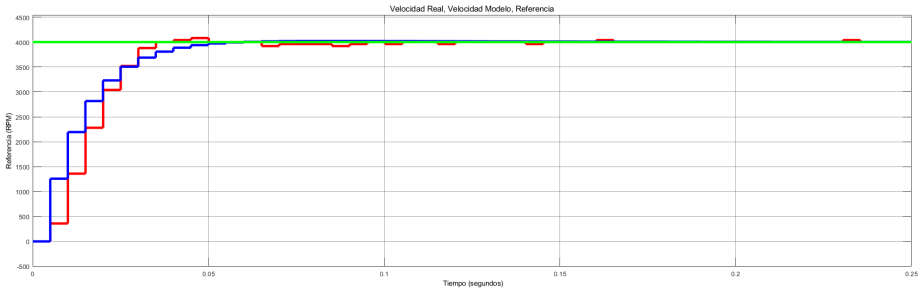

Figura 15: Respuesta al escalón a lazo cerrado

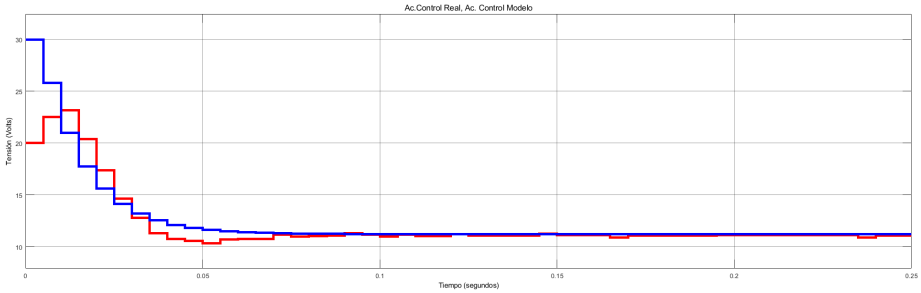

Figura 16: Acción de control

4. Conclusiones

Caracterización de una serie de tanques interconectados: Como conclusión de esta actividad se observa como las *s-function* proporcionan una herramienta sencilla y a la vez poderosa para la simulación de sistemas, dado a que permiten programar de manera sencilla, tanto modelos lineales como no lineales, en el contexto del control basado en estados. El hecho de que la programación de las mismas se basen en estados y salidas hace que la conversión del modelo matemático a un archivo con código ejecutable sea casi directa. Por supuesto, para el éxito de lo anterior, siempre es necesario contar con el *solver* adecuado, que interactúe con nuestros bloques programados de esta manera.

Identificación y control de una planta relevada: En base a lo ensayado es posible concluir que la identificación de la planta, si bien no fue muy complicada de efectuar intentando asemejar la planta propuesta con la respuesta al escalón de la planta *real*, quizás no fue la mejor para luego controlar la planta a partir de métodos empíricos como *Ziegler-Nichols*. Esto se debe a que dichos métodos requieren que la planta identificada cumpla ciertas condiciones especificas para poder ser utilizados. Somos realistas que de haber podido medir el retardo temporal, el método de *Ziegler-Nichols* para plantas de primer orden habría sido sencillo de utilizar y cómodo pero lamentablemente no fue posible.

Quizás, para la utilización de los métodos de *Ziegler-Nichols* hubiera sido mas beneficioso la identificación de la planta mediante una transferencia de mayor orden, no obstante dado que el error cuadrático medio obtenido fue muy pequeño, y que es posible observar en la comparación de las respuestas al escalón son muy similares, se pensó que una identificación de mayor orden brindaría mayor complejidad sin ofrecer mayor precisión.

Se concluye que quizás el *approach* de la utilización de *Ziegler-Nichols* no fue el mejor dado el carácter de la identificación realizada y que quizás se podría haber optado por la utilización de otro método empírico o bien la utilización de algún enfoque teórico para la obtención de un controlador PID como podría haber sido los métodos en frecuencia o en espacio de estados.

Referencias

[1] K. Astrom, R. Murray, "Feedback Systems: An Introduction for Scientists and Engineers", Princeton University Press, Version v2.10b, February 22, 2009