



Facultad de Ingeniería  
Universidad de Buenos Aires

Laboratorio de control automático (86.22)

Trabajo Práctico N°1

S-Functions

2<sup>do</sup> Cuatrimestre, 2019

---

Bergler, Martín	98028	martin.bergler.es2a@gmail.com
Neumarkt Fernández, Leonardo	97471	leoneu928@gmail.com

---

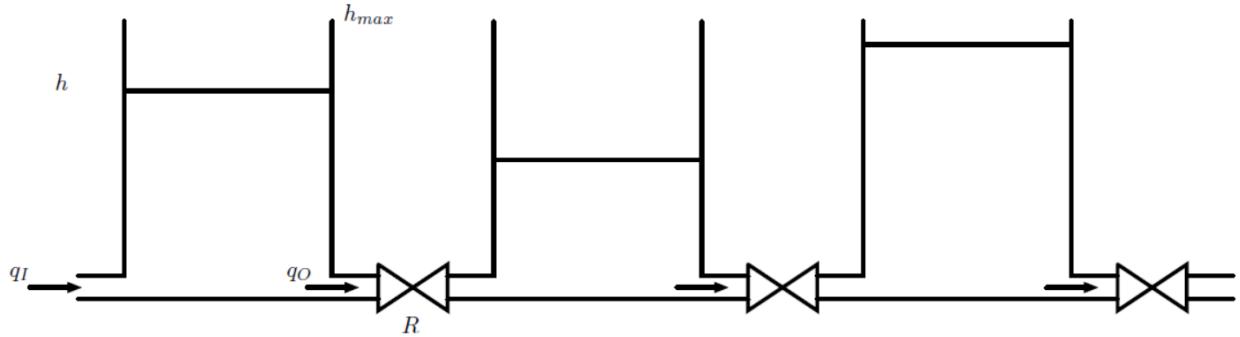
# Índice

<b>1. Enunciado</b>	<b>2</b>
1.1. Ejercicio 1 . . . . .	2
1.2. Ejercicio 2 . . . . .	3
1.3. Ejercicio 3 . . . . .	3
<b>2. Introducción</b>	<b>3</b>
<b>3. Modelo</b>	<b>4</b>
3.1. Ecuaciones matemáticas . . . . .	4
<b>4. Implementación del modelo del Tanque</b>	<b>4</b>
4.1. Validación del modelo . . . . .	6
4.2. Casos de prueba . . . . .	6
4.2.1. 1 Tanque - Caso 1 . . . . .	6
4.2.2. 1 Tanque - Caso 2 . . . . .	7
4.2.3. 1 Tanque - Caso 3 . . . . .	8
4.2.4. 3 Tanques - Caso 1 . . . . .	8
4.2.5. 3 Tanques - Caso 2 . . . . .	10
4.2.6. 3 Tanques - Caso 3 . . . . .	11
4.2.7. 3 Tanques - Caso 4 . . . . .	12
4.2.8. 3 Tanques - Caso 5 . . . . .	12
<b>5. Identificación de un motor de corriente continua</b>	<b>13</b>
5.1. Simulación . . . . .	14
5.2. Validación del modelo . . . . .	16
<b>6. Implementación del controlador PID</b>	<b>17</b>
6.1. Simulación . . . . .	18
6.2. Ajuste del Controlador PID . . . . .	19
<b>7. Ensayos en lazo cerrado</b>	<b>20</b>
<b>8. Conclusiones</b>	<b>21</b>
<b>A. Código Implementado</b>	<b>22</b>
A.1. Ejercicio 1: <b>S-function</b> del tanque . . . . .	22
A.2. Ejercicio 2: <b>S-function</b> del motor . . . . .	27
A.3. Ejercicio 3: <b>S-function</b> del PID . . . . .	32

## 1. Enunciado

### 1.1. Ejercicio 1

Realizar una **S-Function** del tipo **M-File** o **C-Mex** que pueda ser utilizada para modelar el siguiente sistema de 3 tanques interconectados:



**Figura 1:** Modelo de tres tanques

La **S-Function** debe representar el modelo no lineal de un único tanque y tener como parámetros:

- La resistencia de salida de caudal  $R$ .
- La altura máxima  $h_{max}$ .
- El área  $A$  del mismo.
- Su nivel inicial  $h_{ini}$

Sus entradas deben ser:

- El caudal volumétrico de ingreso  $q_i$  ( $u[0]$ ).
- El nivel de slaida  $h_0$  ( $u[1]$ ).

Las salidas deben ser:

- El nivel del tanque  $h$  ( $y[0]$ ).
- El caudal volumétrico de salida  $q_0$  ( $y[1]$ ).

La función deberá respetar la siguiente interfaz:

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

Considerar que el caudal volumétrico de salida de un tanque responde al principio de Bernoulli, que se puede simplificar de la siguiente manera:

$$q_{O_i} = \frac{1}{R} \sqrt{\Delta h_{i+1,i}} \quad (1)$$

Es decir, es inversamente proporcional a la resistencia de salida y directamente proporcional a la raíz de la diferencia de alturas con el tanque siguiente. Su sentido dependerá de  $\Delta h_{i+1,i}$ .

## 1.2. Ejercicio 2

Realizar la identificación de un motor de corriente continua. Se cuenta con una **S-Function M-File**:

```
function [sys,x0,str,ts] = dcmotor_qenc(t,x,u,flag,encoder_ppv)
```

que representa el sistema real bajo estudio sobre el cuál se realizará el experimento de identificación.

Posee las siguiente características:

- Entrada en tensión **u[0]** y se sabe que su valor nominal es de 12 V.
- Salidas de medición de un encoder en cuadratura (**y[0]** e **y[1]**) montado al eje, que genera **encoder\_ppv** pulsos por vuelta.

## 1.3. Ejercicio 3

Utilizar el modelo identificado del ejercicio 2 para cerrar un lazo de control **PID** de velocidad similar al del ítem anterior pero en tiempo discreto. Ajustar el control mediante prueba y error hasta lograr un rendimiento deseado.

Aplique el controlador **PID** obtenido sobre el modelo con **S-Function** original dado en el ejercicio 2. Compare el rendimiento en base al mismo criterio.

## 2. Introducción

En esta sección se muestra una introducción teórica a los temas involucrados en la realización del Trabajo Práctico:

- **S-Functions:** Las *S-Functions* sirven para extender las capacidades de *Simulink* y con ellas creas bloques a ser utilizados por dicho software. En definitiva una *S-Function* es una descripción en nuestro caso en lenguaje *MATLAB* (*M-File*) de un bloque de *Simulink*. Además es posible programarlo en lenguaje C (*C-Mex*).
  - **M-File:** Son archivos de extensión .m, que pueden ser *scripts* en lo que se indican comandos que se quieran ejecutar en cierto orden y que utilizan funciones propias de *MATLAB*. Con un *M-File* se indica la dinámica que debe tener un bloque de *S-Function* que será utilizado en *Simulink*.
  - **C-Mex:** Es un tipo de archivo que provee una interfase de comunicación entre *MATLAB* y funciones escritas en lenguaje de programación C. Estos son de extensión .c y pueden ser utilizados por un bloque de *S-Function*.
- **Solver:** Se llama *Solver* al método aplicado por *Simulink* para resolver un conjunto de ecuaciones diferenciales, las cuales representan un modelo. En el simulador se proveen varios *Solvers*. El tipo a utilizar dependerá de la simulación a realizar y se deberá tener en cuenta factores como el tipo de paso (fijo o variable), la dinámica del sistema, la mínima resolución que necesitamos para el problema en cuestión, la velocidad de cómputo, entre otros.
- **Modelo del tanque:** Se modeló a los tanques mediante diferentes parámetros como su altura máxima, área y la resistencia de caudal de entrada y salida. A su vez se definió un parámetro para indicar la altura inicial de líquido que contiene cada tanque. Además los incrementos o decrementos de alturas de cada uno de los tanques depende de sus caudales de entrada y salida. Estos últimos a su vez dependen de las alturas de los tanques adyacentes.
- **Identificación del motor:** Se recibió un bloque correspondiente a una *S-Function*, la cuál devuelve pulsos leídos de un motor que tiene una superficie circular partida en dos colores distintos. Para realizar la identificación se contaron los pulsos en cierta ventana de tiempo y luego se hizo la conversión a RPM para obtener la velocidad del motor.

- Control PID: Un controlador PID (Proporcional, Integral, Derivativo) funciona mediante realimentación y mide y corrige la desviación a la salida de un sistema comparándola con una señal de referencia. El término proporcional sirve para que el error en estado estacionario tienda a cero. Esta constante controla los sobre picos que tiene el sistema frente a una excitación. A diferencia del otros dos términos, este no tiene dependencia temporal.

Luego se tiene el termino integral que busca eliminar el error en estado estacionario que no pudo ser corregido por el control proporcional. Lo que realiza el control integral es integrar el error para poder promediarlo, luego se multiplica por una constante  $K_i$  y finalmente se suma al proporcional. Por último tenemos el término derivativo, el cuál juega un rol importante cuando aparece una variación en el valor del error. En este caso se deriva la señal de error respecto del tiempo, luego se la multiplica por una constante  $K_d$  y finalmente se suma a los dos casos anteriores para formar el control propiamente dicho PID.

### 3. Modelo

#### 3.1. Ecuaciones matemáticas

La altura de un tanque de agua puede obtenerse mediante la ecuación (2).

$$h(t) = h(0) + \int_0^T \frac{Q_i - Q_o}{A} dt \quad (2)$$

Donde  $h(0)$  es la altura inicial del tanque,  $Q_i$  y  $Q_o$  el caudal de entrada y salida, respectivamente, y  $A$  el área del tanque.

Luego, para realizar la dinámica del modelo se utilizo la derivada de la altura, la cuál se obtiene de la ecuación (2). Esta se muestra en la ecuación (3).

$$\dot{h}(t) = \frac{Q_i - Q_o}{A} \quad (3)$$

El caudal de salida se describió por medio de la ecuación (4).

$$Q_o = \frac{1}{R} \sqrt{h - h_o} \quad (4)$$

Donde  $h_o$  es la altura de agua del siguiente tanque.

### 4. Implementación del modelo del Tanque

En esta sección se explica la implementación del modelo del tanque mediante una *S-Function*. Para realizarlo se utilizó una plantilla de *S-Function* incorporada en *MATLAB*, la cuál se puede visualizar utilizando el comando `open('sfuntmpl.m')`. Cabe aclarar que la presente función es para un único tanque y luego, mediante el correspondiente bloque de *Simulink*, se simularán la cantidad de tanques correspondientes.

A continuación se muestra la función para la inicialización (`mdlInitializeSizes()`), en donde se definieron 2 entradas ( $u$ ), las cuales representan el caudal de entrada y la altura del tanque siguiente y 2 salidas ( $y$ ), correspondientes a la altura del tanque y el caudal de salida. Además se definió que si la altura inicial del tanque es mayor a la altura máxima ( $h_{max}$ ), la condición inicial debía ser esa altura máxima, lo cuál representa que el tanque desbordó. Análogamente se tuvo en cuenta si la altura es menor a la altura mínima (0). Estos casos se tuvieron en cuenta para que el modelo se aproxime de manera más certera al sistema real.

```

function [sys ,x0 ,str ,ts ,simStateCompliance]=mdlInitializeSizes(hmax,hini)
sizes = simsizes;
sizes.NumContStates = 1;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % at least one sample time is needed
sys = simsizes(sizes);
% initialize the initial conditions
if hini>=hmax
    x0=hmax; % El tanque llego a su nivel maximo y desborda.
elseif hini<=0
    x0=0; % Tanque vacio
else
    x0=hini;
end

```

Luego se codificó la función *mdlDerivatives()*, la cuál hace uso de la ecuación (3). Se contemplaron 3 casos. Por un lado, si la altura supera el valor seteado como máximo, es decir  $h_{max}$ , como caudal de salida se utiliza la ecuación (4), reemplazando  $h$  por  $h_{max}$ . También se tiene en cuenta el signo de la diferencia de alturas ( $h_{max} - u(2)$ ) para contemplar si el caudal es saliente o entrante al tanque. El siguiente caso contempla si el estado de la altura es menor a 0, la cuál representa la altura mínima del tanque. Entonces se realiza una operación similar a la ya mencionada, reemplazando la altura por 0. Estos dos casos explicados contemplan entonces los casos en que el tanque se está llenando de más y si el tanque se encuentra vacío. Finalmente, el tercer caso, contempla si el estado de la altura se encuentra entre medio de los límites inferior y superior del tanque.

```

function sys=mdlDerivatives(t ,x,u,R,hmax,A, hini)
if x>=hmax %tanque lleno
    h_derivada = (1/A)*(u(1)-(1/R)*sqrt(abs(hmax-u(2)))*sign(hmax-u(2)));
    %Se usa sign por si el caudal va en un sentido o en el otro
    if h_derivada >= 0 %se esta llenando de mas
        sys=0;
    else
        sys=h_derivada;
    end
elseif x<=0 % el tanque esta vacio
    h_derivada = (1/A)*(u(1)-(1/R)*sqrt(abs(0-u(2)))*sign(0-u(2)));
    if h_derivada <= 0 %se esta vaciando de mas
        sys=0;
    else
        sys=h_derivada;
    end
else
    sys = (1/A)*(u(1)-(1/R)*sqrt(abs(x-u(2)))*sign(x-u(2)));
end

```

Por último tenemos la función *mdlOutputs()*. Nuevamente se contemplan los tres casos ya explicados anteriormente. Las salidas se definieron como  $y(1) = h$  e  $y(2) = q_o$ , siendo  $h$  la altura del tanque y  $q_o$  el caudal de

salida del mismo. Si el estado de la altura actual supera la altura máxima, la altura será este último valor y el caudal será definido como se explicó para el caso de la función *mdlDerivatives()*. Análogamente se realizó la definición de la salida si el estado actual es menor a 0. Por último si la altura actual cumple con los límites, la altura del tanque será esa misma y el caudal de salida coincidirá con el de la ecuación (4).

```
function sys=mdlOutputs(t,x,u,R,hmax,A,hini)
if x(1)>=hmax
    sys=[hmax (1/R)*sqrt ( abs(hmax-u(2)))*sign (hmax-u(2)) ];
elseif x(1)<=0
    sys=[0 (1/R)*sqrt ( abs(0-u(2)))*sign (0-u(2)) ];
else
    sys=[x(1) (1/R)*sqrt ( abs(x(1)-u(2)))*sign (x(1)-u(2)) ];
end
```

Las demás funciones contenidas en la plantilla de la *S-Function* se mantuvieron sin modificaciones.

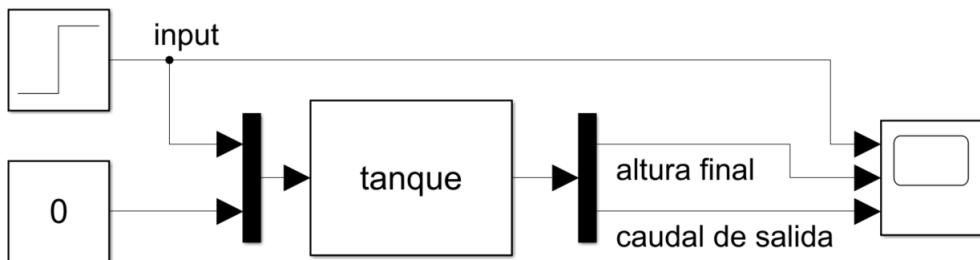
## 4.1. Validación del modelo

Para la validación del modelo se realizó la simulación de un único tanque y de tres tanques conectados en serie, con una entrada escalón. Se configuró el *solver* en **ode45**, de paso variable y tolerancia relativa de  $10^{-4}$ . El resto de los parámetros se dejaron en auto.

## 4.2. Casos de prueba

### 4.2.1. 1 Tanque - Caso 1

El primer caso de prueba que se llevó a cabo es el de un único tanque. En la figura 2 se muestra el esquema realizado en *Simulink*. A la entrada se colocó un multiplexador para poner por un lado un escalón representando el caudal de entrada del tanque y por otro lado la altura del tanque siguiente, que en este caso es 0, ya que el sistema es de un único tanque.



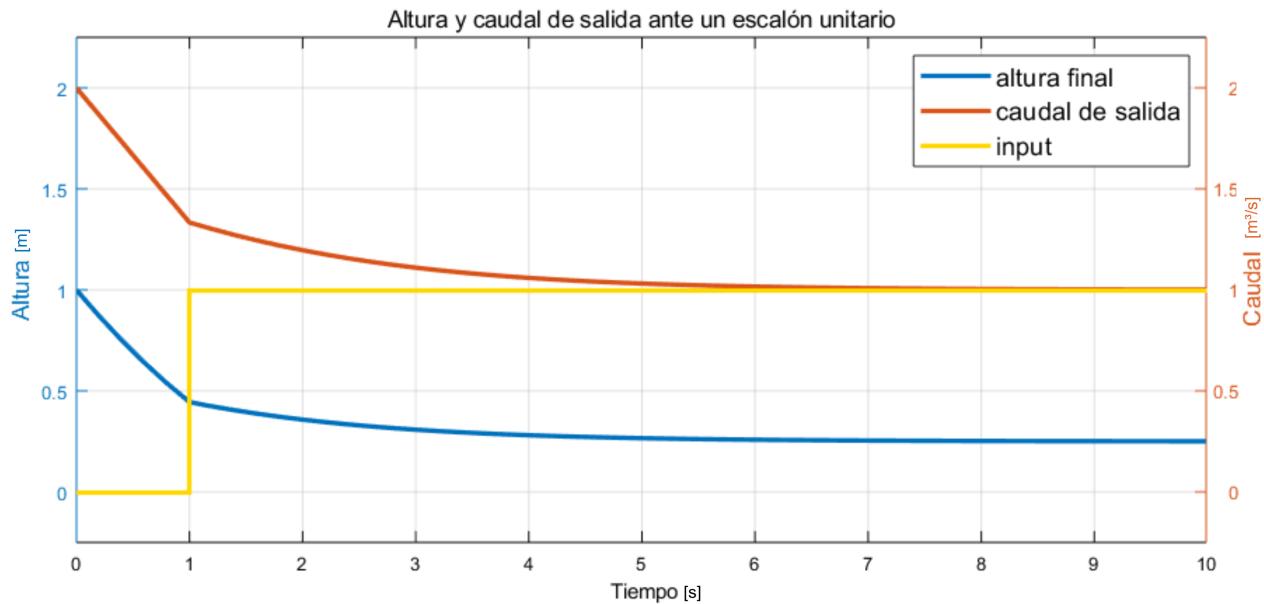
**Figura 2:** Simulink de un tanque.

Los parámetros utilizados para este tanque se muestran en la tabla (1).

R [s/m <sup>5/2</sup> ]	h <sub>max</sub> [m <sup>2</sup> ]	A [m <sup>2</sup> ]	h <sub>ini</sub> [m]
0.5	10	3	1

**Tabla 1:** Parámetros correspondientes a la simulación de 1 tanque.

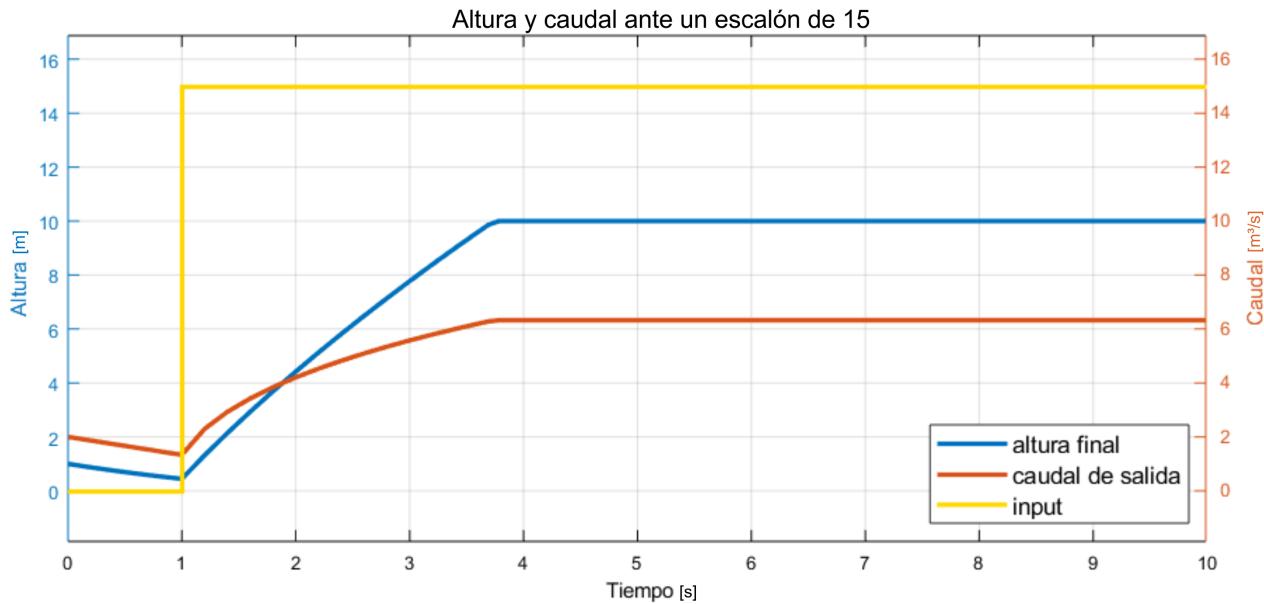
En la figura 3 se muestra el resultado de la simulación correspondiente a la figura 2. Se colocó como caudal de entrada un escalón de valor final  $1 \text{ m}^3/\text{s}$ . Como se puede observar el tanque empieza con una altura inicial de  $1 \text{ m}$ , la cuál va disminuyendo inicialmente, ya que comienza a entrar agua al tanque pasado 1 segundo. Una vez que empieza a entrar agua en el tanque la altura final decrece con menor pendiente hasta establecerse en un valor final de  $0,25 \text{ m}$ . El caudal de salida presenta un comportamiento similar en cuanto a su forma y se establece en el mismo valor que el caudal de entrada.



**Figura 3:** Simulación de un tanque

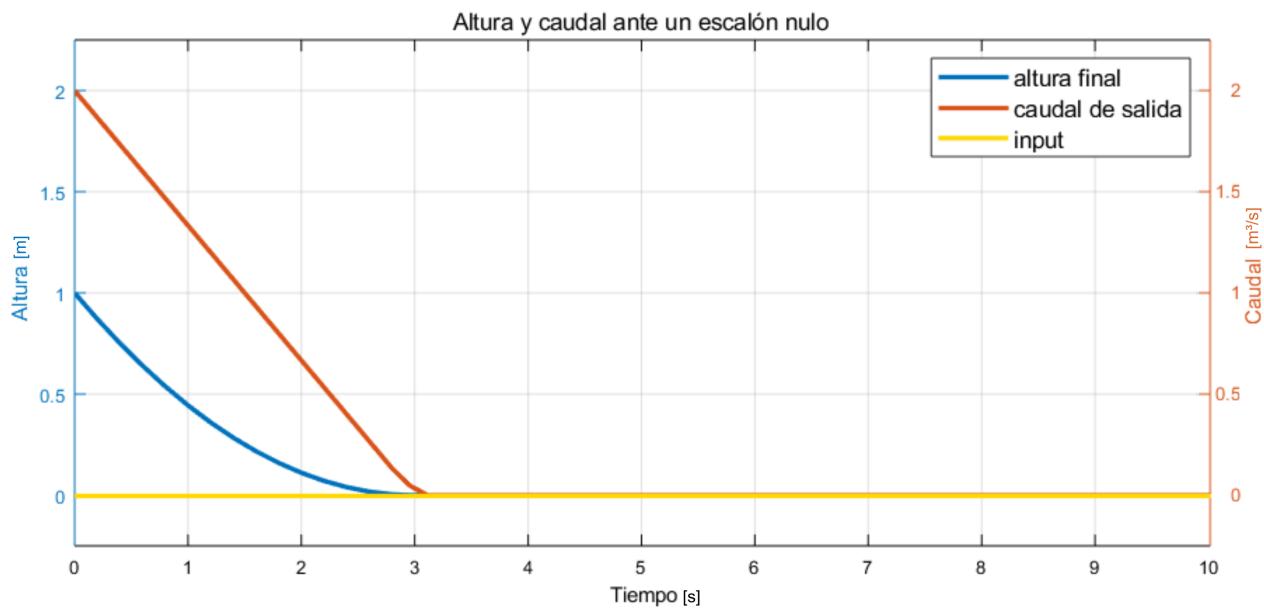
#### 4.2.2. 1 Tanque - Caso 2

Se utilizó el mismo diagrama que el de la figura 2, pero cambiando la entrada a un valor de caudal final de  $15 \text{ m}^3/\text{s}$  y manteniendo los parámetros en los mismos valores. Dicha simulación se muestra en la figura 4. Inicialmente el comportamiento del sistema es idéntico al de la simulación anterior, como era de esperarse, pero una vez que ingresa agua al tanque, este comienza a llenarse, dado que el caudal de entrada es considerablemente mayor al de la simulación anterior. Finalmente la altura del tanque alcanza su máximo, el cuál fue configurado en  $10 \text{ m}$ .

**Figura 4:** Simulación de un tanque

#### 4.2.3. 1 Tanque - Caso 3

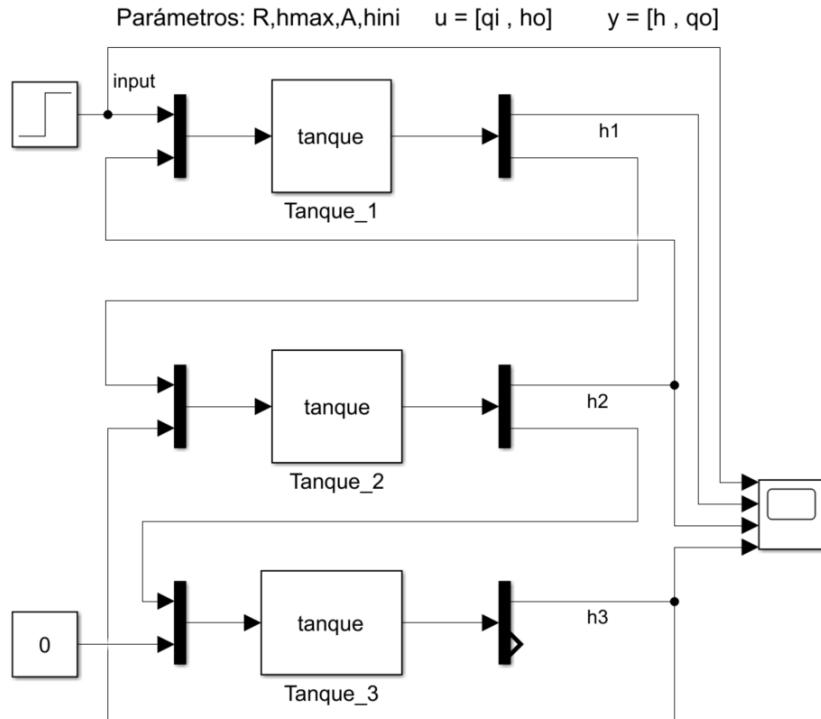
Como última prueba para un tanque, se modificó la entrada a un valor de caudal nulo y manteniendo los parámetros en los mismos valores. Dicha simulación se muestra en la figura 5. En este caso el tanque se descarga tan rápido como le permite la resistencia de salida de caudal, hasta que llega a un valor de altura y caudal de salida nulo.

**Figura 5:** Simulación de un tanque

#### 4.2.4. 3 Tanques - Caso 1

Luego se realizó la simulación para tres tanques colocados en serie, donde se excita al sistema mediante un escalón como caudal de entrada del primer tanque. El caudal de entrada de los dos tanques siguientes se

corresponde con el de salida del correspondiente tanque anterior. Además como entrada de los tanques se toma la altura del tanque siguiente, salvo en el último, donde este valor es 0. En la figura 6 se muestra el diagrama realizado con *Simulink* para simular el sistema.



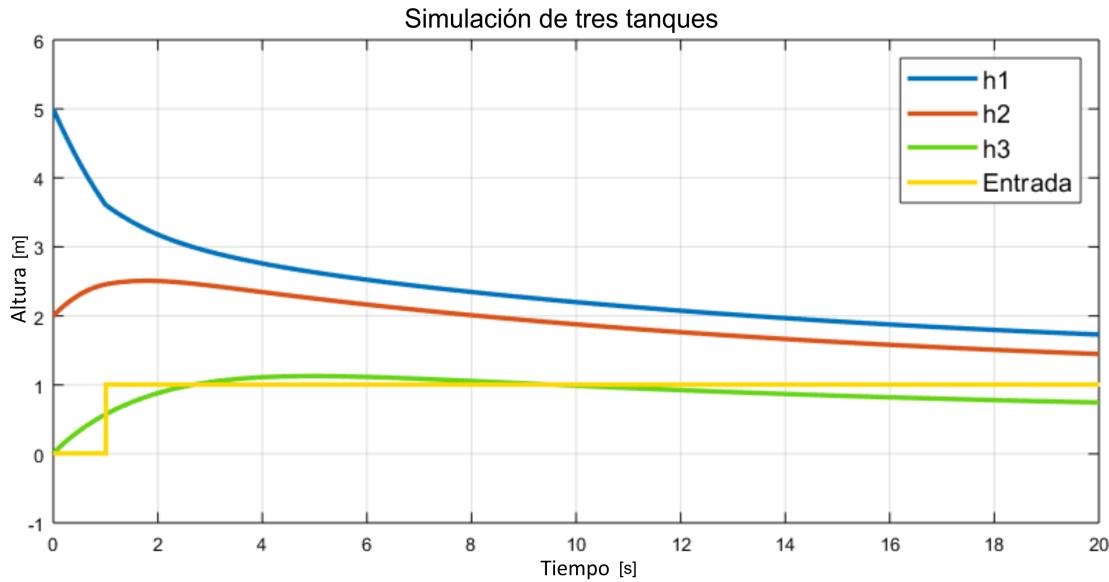
**Figura 6:** *Simulink* de tres tanques.

Los parámetros que se utilizaron para cada tanque se indican en la tabla 2.

Tanque	$R$ [s/m <sup>5/2</sup> ]	$h_{max}$ [m <sup>2</sup> ]	$A$ [m <sup>2</sup> ]	$h_{ini}$ [m]
1	0.50	10	2	5
2	0.75	10	2	2
3	0.75	6	2	0

**Tabla 2:** Parámetros correspondientes a la simulación de 3 tanques.

En la figura 2 se puede observar el comportamiento del sistema. Inicialmente, al no tener caudal de entrada en el primer tanque, este se descarga, mientras que los otros dos incrementan su altura de agua. Luego, cuando comienza a ingresar agua en el primer tanque, este se vacía más lentamente. Los otros dos tanques siguen un comportamiento similar. Los valores de resistencia del caudal de salida y el área de los tanques no son de valores lo suficientemente altos como para contrarrestar el decremento de altura del agua, por lo que los tres tanques se vaciarán hasta cierta altura, en donde quedará un valor constante.

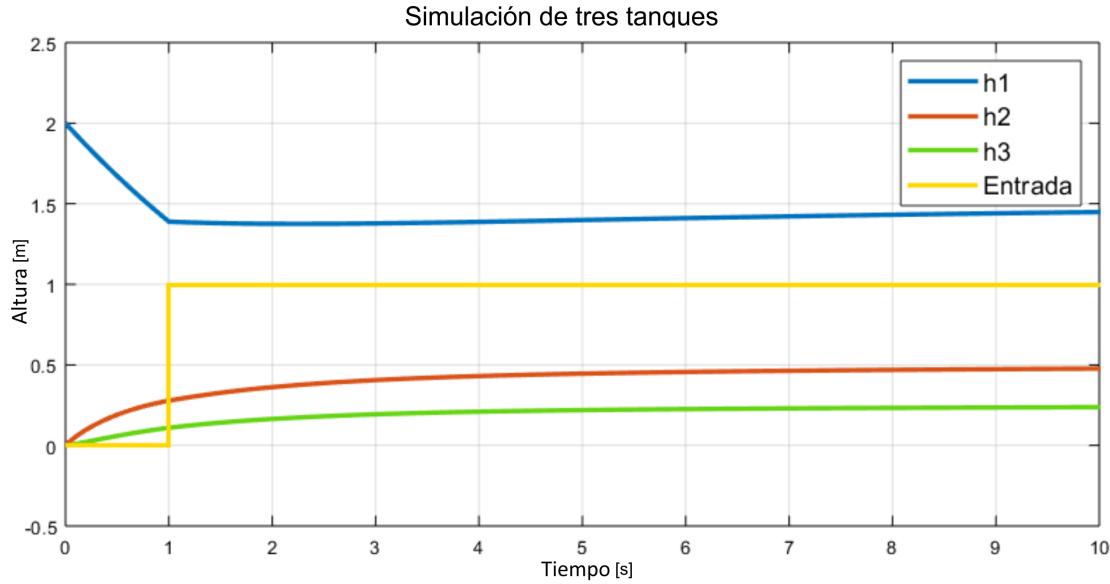
**Figura 7:** Simulación de tres tanques**4.2.5. 3 Tanques - Caso 2**

Luego se variaron los parámetros de los tanques, como se puede observar en la figura 3.

Tanque	R [s/m <sup>5/2</sup> ]	h <sub>max</sub> [m <sup>2</sup> ]	A [m <sup>2</sup> ]	h <sub>ini</sub> [m]
1	1	10	2	2
2	0.5	10	2	0
3	0.5	8	2	0

**Tabla 3:** Parámetros correspondientes a la simulación de 3 tanques.

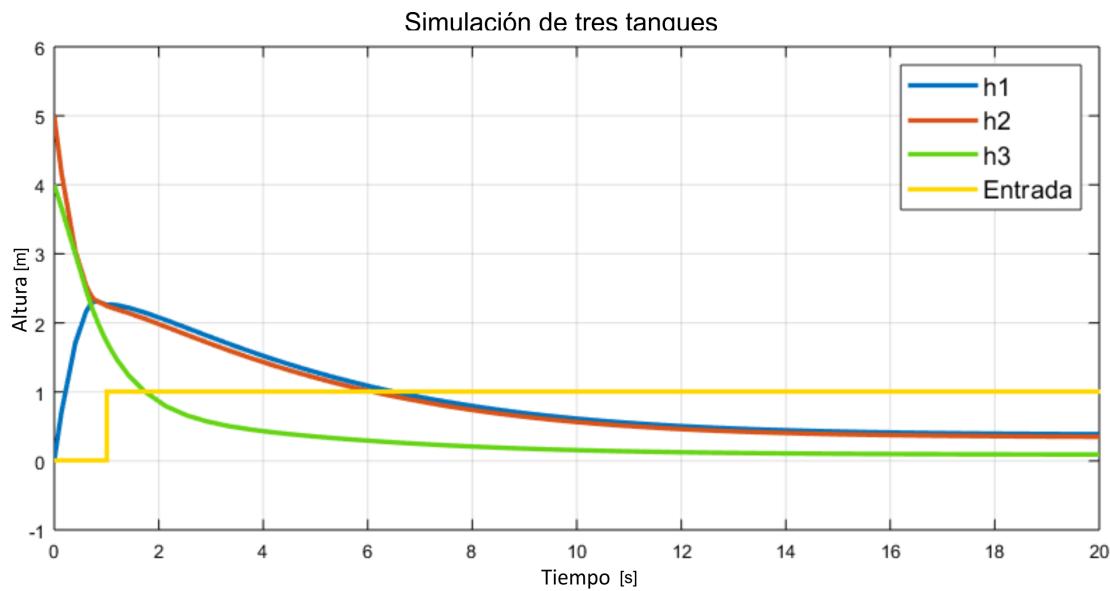
La simulación de los tres tanques con los parámetros de la tabla 3 se muestran en la figura 3. Al elevar la resistencia de caudal del primer tanque, este se llenará a partir del momento en que ingresa agua como entrada. Por otro lado, si bien la resistencia de los dos tanques siguientes es menor al caso anterior, como el primer tanque no llega a vaciarse, estos tendrán un caudal de entrada que no dejará que la altura del agua en ellos disminuya.

**Figura 8:** Simulación de tres tanques

#### 4.2.6. 3 Tanques - Caso 3

Bajando considerablemente el valor de la resistencia de caudal de salida en los tres tanques, como se indica en la tabla 4, puede observarse en la figura 5 que la altura de los tanques disminuye inmediatamente con gran pendiente.

Tanque	R [s/m <sup>5/2</sup> ]	$h_{max}$ [m <sup>2</sup> ]	A [m <sup>2</sup> ]	$h_{ini}$ [m]
1	0,2	10	2	0
2	0,5	10	2	5
3	0,3	4	2	4

**Tabla 4:** Parámetros correspondientes a la simulación de 3 tanques.**Figura 9:** Simulación de tres tanques

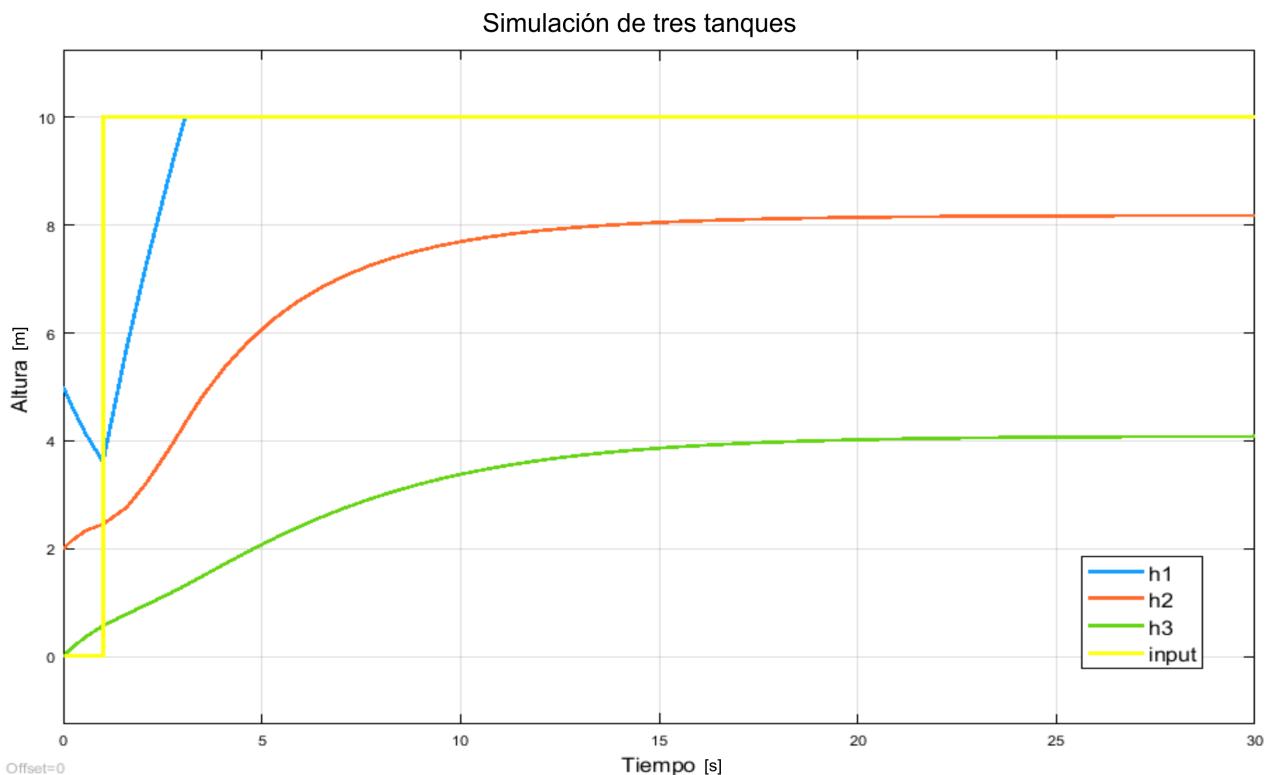
#### 4.2.7. 3 Tanques - Caso 4

Luego se cambió la señal de entrada, colocando un escalón de caudal a la entrada tal que la altura del primer tanque sature. En la tabla 5 se muestran los parámetros configurados para este caso.

Tanque	$R$ [s/m $^{5/2}$ ]	$h_{max}$ [m $^2$ ]	A [m $^2$ ]	$h_{ini}$ [m]
1	0,5	10	2	5
2	0,75	10	2	2
3	0,75	6	2	0

**Tabla 5:** Parámetros correspondientes a la simulación de 3 tanques.

En la figura 10 se muestra la respuesta del sistema. Como se puede ver, inicialmente el tanque 1 se descarga, ya que no tiene caudal de entrada, haciendo que los otros dos tanques vayan llenándose. Una vez que el tanque 1 recibe caudal a su entrada, la altura de este se eleva rápidamente hasta alcanzar su punto máximo. Los otros dos tanques también elevan su altura, pero con menor pendiente.



**Figura 10:** Simulación de tres tanques

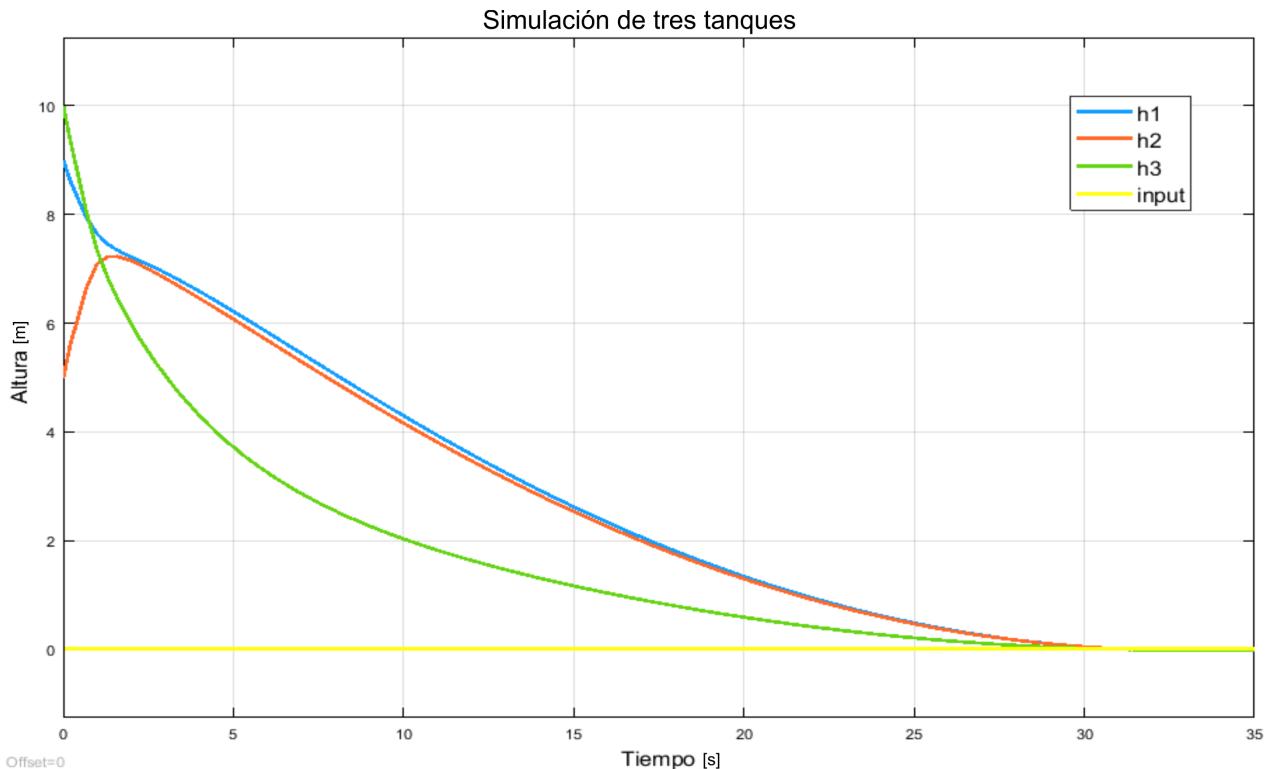
#### 4.2.8. 3 Tanques - Caso 5

Por último se llevó a cabo una simulación sin caudal de entrada, definiendo alturas iniciales para cada tanque. Los parámetros utilizados se muestran en la tabla 6.

Tanque	R [s/m <sup>5/2</sup> ]	$h_{max}$ [m <sup>2</sup> ]	A [m <sup>2</sup> ]	$h_{ini}$ [m]
1	0,5	10	2	9
2	1	10	2	5
3	0,75	10	2	10

**Tabla 6:** Parámetros correspondientes a la simulación de 3 tanques.

En la figura 11 se observa como la altura de todos los tanques llega al valor de 0 m, al no haber ningún caudal de entrada del sistema.

**Figura 11:** Simulación de tres tanques

## 5. Identificación de un motor de corriente continua

Para realizar la identificación de un motor de corriente continua se observó el comportamiento de un encoder de cuadratura. Posee dos salidas A y B, correspondiente a dos sensores, dependiendo cuál de los dos se registre primero, se podrá determinar el sentido de giro del motor.

El modelo del motor esta dado por **dcmotor\_qenc.p**. Se realizó la siguiente *S-function*, la cual se encargará de determinar la velocidad del motor, medida en revoluciones por minuto (*RPM*). Recibe como parámetros el tiempo de muestreo (*Ts*), la cantidad de pulsos por vuelta (*ppv*) y el ancho de la ventana de tiempo en el que se detectan los pulsos (*Tw*).

Se toma como entrada en escalón de tensión  $u[0]$  y de valor nominal de 12 V. Se consideran como salidas de medición del *encoder* en cuadratura ( $y[0]$  e  $y[1]$ ) montado al eje, que genera *encoder\_ppv* pulsos por vuelta.

```

function [ sys ,x0,str ,ts ,simStateCompliance] = dcmotor_speed(t ,x ,u ,flag ,Ts ,ppv ,Tw)
[...]
function sys=mdlUpdate(t ,x ,u ,ppv ,Tw)

% x = [ estado anterior , contador , tiempo inicial de ventana , velocidad]
% u = [ tension de entrada]

if x(2) == 0
    x(3) = t;
end

if u > x(1)
    x(2) = x(2)+1; % Se encontro un pulso , por lo que se aumenta el contador
end

if (t-x(3)) > Tw % Se paso la ventana de tiempo
    % Defino la velocidad a partir de la cantidad de pulsos contados:
    x(4) = (60*x(2))/(ppv*Tw); % Escribo a la velocidad en RPM
    x(2) = 0; % Se inicializa el contador en 0 para la proxima ventana de tiempo
end

x(1)=u;

sys = x;

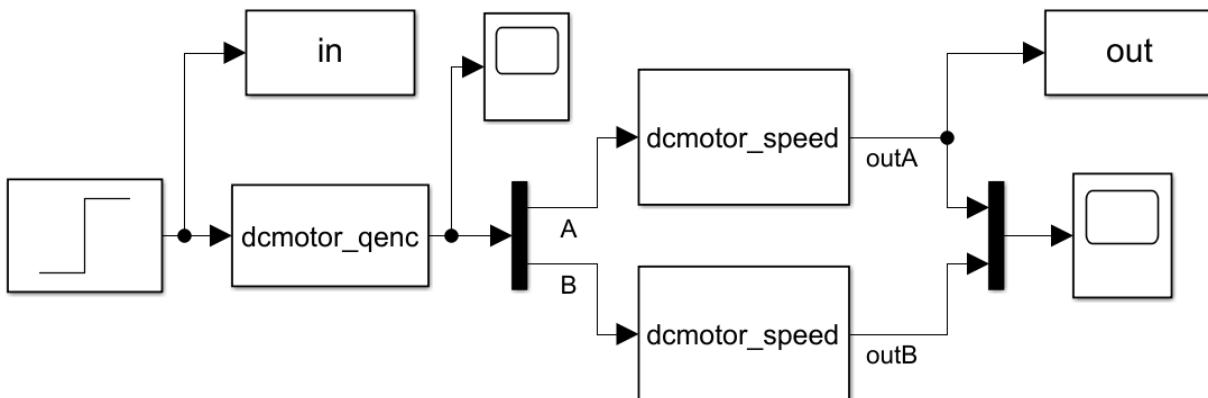
% end mdlUpdate

```

Si el contador ( $x(2)$ ) se encuentra en 0, se asigna el tiempo actual al tiempo inicial de la ventana ( $x(3)$ ). Luego compara la entrada con el estado anterior ( $x(1)$ ) y en caso que la entrada sea mayor, es decir que se detecte un pulso, se incrementa el contador. Luego se compara si el tiempo transcurrido es mayor al ancho de la ventana, y en caso afirmativo se realiza la conversión de la cantidad de pulsos a revoluciones por minuto ( $x(4)$ ), y se resetea el contador. Por último se asigna el estado actual como estado anterior ( $x(1)$ ).

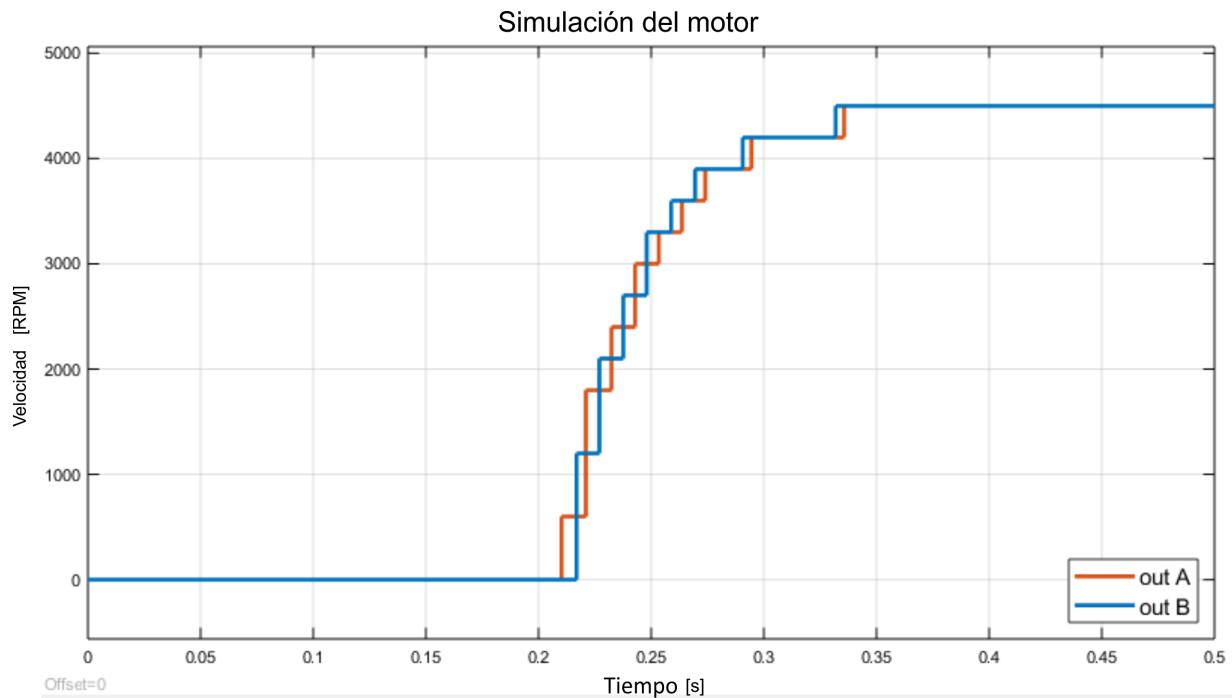
## 5.1. Simulación

A continuación se realizó la simulación en *Simulink*, aplicando un escalón de tensión de 12 V.

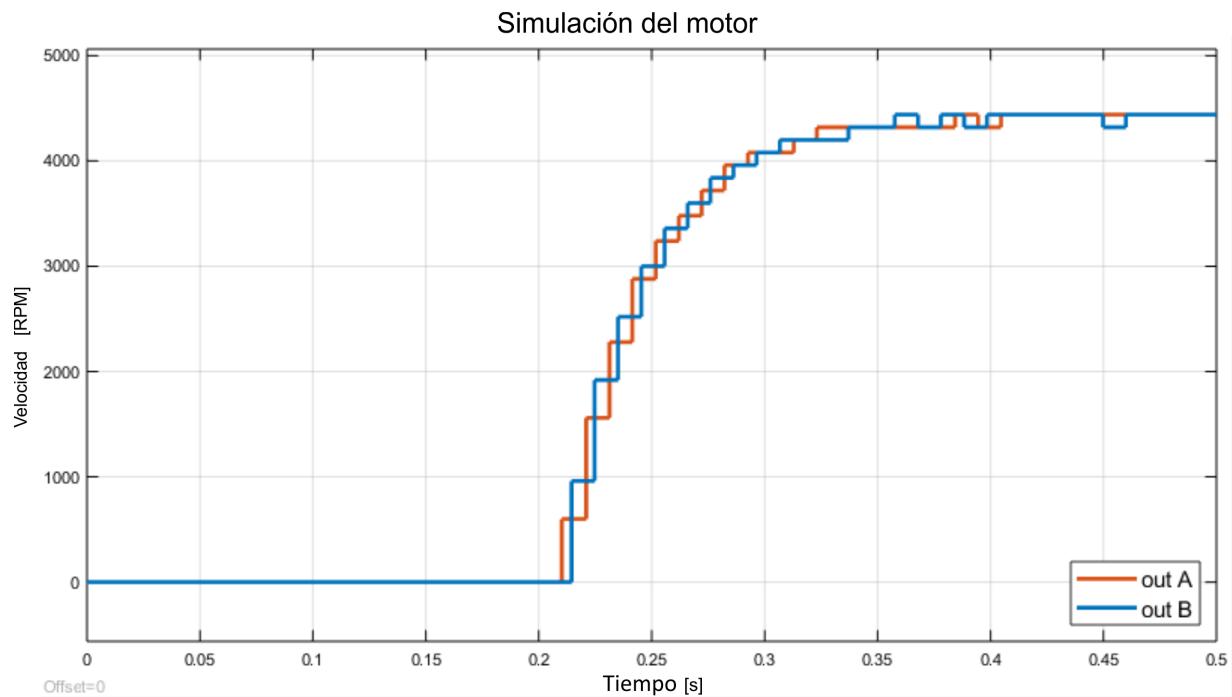


**Figura 12:** Simulink del motor

Para la primera simulación (Figura 13), se establecieron los siguientes parámetros:  $Ts=0,0001$ ,  $ppv=20$  y  $Tw=0,01$ .

**Figura 13:** Simulación del motor con ppv=20

Luego se incrementó el valor de ppv a 50.

**Figura 14:** Simulación del motor con ppv=50

Se puede apreciar que si bien la velocidad final alcanzada es la misma, 4500 rpm, en el segundo caso se registra una mayor cantidad de vueltas o revoluciones a lo largo del tiempo, manteniendo el mismo ancho de la

ventana.

## 5.2. Validación del modelo

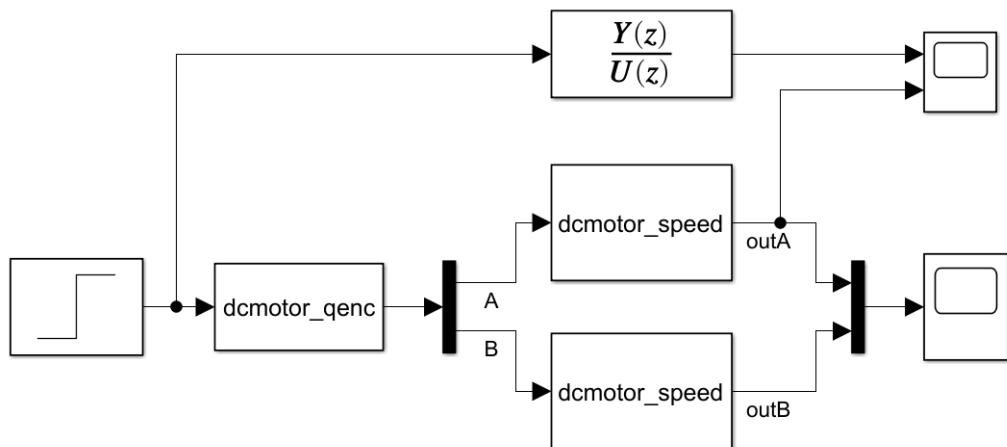
Para obtener la validación del modelo del motor analizado en el caso anterior, se utilizó la función de *MatLab arx()* (*Autoregressive with Exogenous Variables*), la cuál aproxima la función con una curva, en este caso de segundo orden, utilizando el método de cuadrados mínimos.

```
order=[2,1,1]; % Se establece el orden del modelo.
data=iddata(out.Data,in.Data,Ts); % Se exportan los datos desde Simulink.
sys=arx(data,order);
```

De este modo se obtiene la siguiente transferencia:

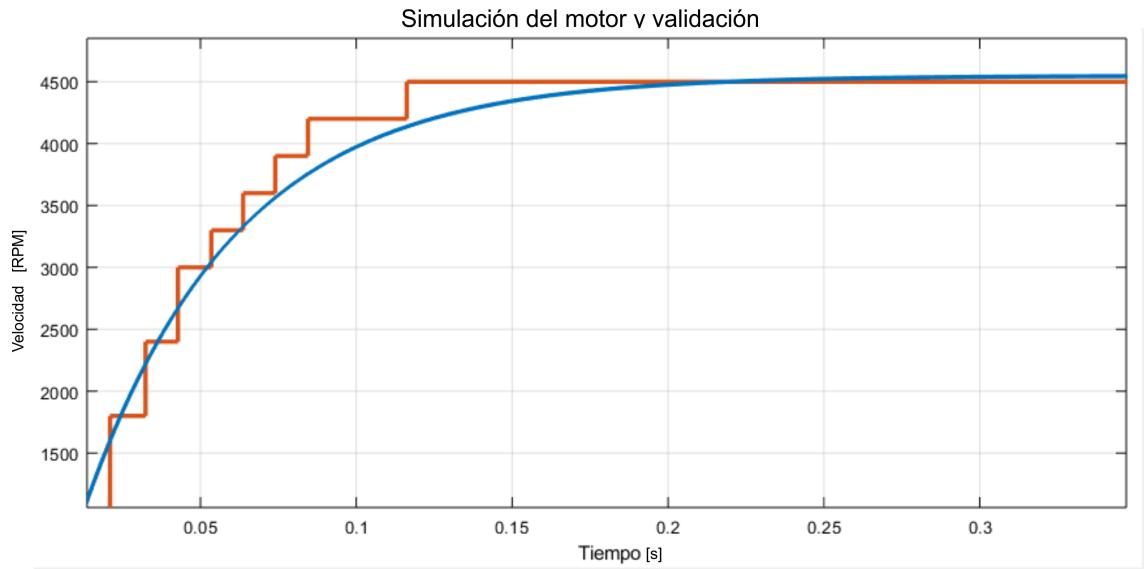
$$H(z) = \frac{0,777z^{-1}}{1 - 0,9918z^{-1} - 0,006138z^{-2}} \quad (5)$$

Se realiza la misma simulación de *Simulink* de la medición del motor y se la gráfica en conjunto con la validación del modelo.



**Figura 15:** Simulink del motor y su validación

En la figura 16 puede observarse que la transferencia hallada se corresponde con la simulación realizada, ya que las formas de las curvas son similares.



**Figura 16:** Simulación del motor y su validación

## 6. Implementación del controlador PID

Se desea controlar el sistema a lazo cerrado. Para ello se utilizó un controlador PID (proporcional, integrador, derivador). Se implementó mediante una *S-function* ya realizada previamente en la guía de ejercicios de la materia.

```

function [sys ,x0 ,str ,ts ,simStateCompliance] = pid_sfunc(t ,x ,u ,flag ,Kp,Kd,Ki ,h ,N)
[...]
function sys=mdlUpdate( t ,x ,u ,Kp ,Kd ,Ki ,h ,N)

gamma=Kd/N;
A=[1 0 0; 0 gamma/(gamma+h) (Kp*Kd)/(gamma+h) ; 0 0 0];
B=[h*Kp*Ki -h*Kp*Ki; 0 -Kp*Kd/(gamma+h); 0 1];
sys = [A*x+B*u];
%end mdlUpdate
[...]
function sys=mdlOutputs( t ,x ,u ,Kp ,Kd ,Ki ,h ,N)

gamma=Kd/N;

C=[1 gamma/(gamma+h) Kp*Kd/(gamma+h) ];
D=[Kp (-Kp-(Kp*Kd/(gamma+h))) ];
sys = [C*x+D*u];
%end mdlOutputs

```

Para realizar la *S-Function* para el PID discreto se partió de la ecuación (6).

$$u_k = P_k + I_k + D_k \quad (6)$$

Donde cada término se define a continuación:

$$P_k = K_p (r_k - y_k) \quad (7)$$

$$I_{k+1} = K_p K_i h \sum_{j=0}^k (r_j - y_j) = I_k + K_p K_i h (r_k - y_k) \quad (8)$$

$$D_k = \frac{\gamma}{\gamma + h} D_{k-1} - \frac{K_p K_d}{\gamma + h} (y_k - y_{k-1}) \quad (9)$$

Las constantes utilizadas en las ecuaciones (7), (8) y (9), que luego serán utilizadas en la implementación de la *S-Function* son  $h$ , que es el período de muestreo,  $r_k$  es la señal de referencia,  $y_k$  es la variable medida,  $K_p$  la constante de proporcionalidad,  $K_i$  la constante de integrativa,  $K_d$  la constante de derivativa y  $N$  el coeficiente del filtro.

Trabajando con estas expresiones se llega a un sistema de ecuaciones, el cuál se puede describir con las matrices de estado.

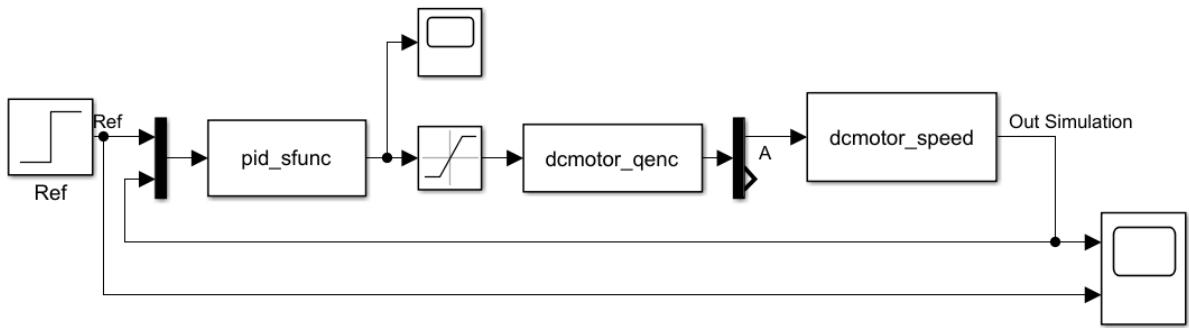
Mediante la *S-Function* se establecen las matrices de estado **A**, **B**, **C** y **D** las cuales determinan el comportamiento del sistema correspondiente al PID. En la función *mdlUpdate* se define la ecuación  $\dot{x} = Ax + Bu$  con las respectivas matrices **A** y **B**. Por otro lado en la función *mdlOutputs* se define la función de salida  $y = Cx + Du$ , junto a las matrices **C** y **D**. Dichas matrices se muestran a continuación.

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{g}{(\gamma+h)} & \frac{K_p * K_d}{(\gamma+h)} \\ 0 & 0 & 0 \end{pmatrix} \quad \mathbf{B} = \begin{pmatrix} h * K_p * K_i & -h * K_p * K_i \\ 0 & \frac{-K_p * K_d}{(\gamma+h)} \\ 0 & 1 \end{pmatrix} \quad (10)$$

$$\mathbf{C} = \begin{pmatrix} 1 & \frac{g}{(\gamma+h)} & \frac{K_p * K_d}{(\gamma+h)} \end{pmatrix} \quad \mathbf{D} = \begin{pmatrix} K_p & \frac{-K_p - K_p * K_d}{(\gamma+h)} \end{pmatrix} \quad (11)$$

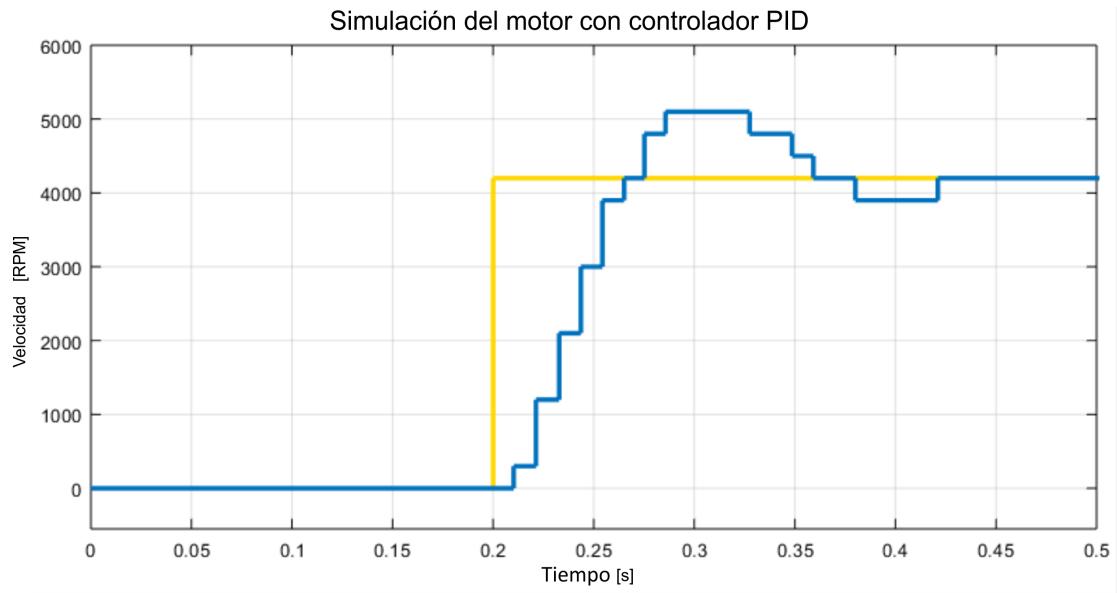
## 6.1. Simulación

Para esta situación se estableció un escalón fijado en un punto de operación de 4200 RPM. El controlador posee un saturador a la salida del mismo, el cual limita los picos de tensión mayores a 30 V y menores a -30 V.



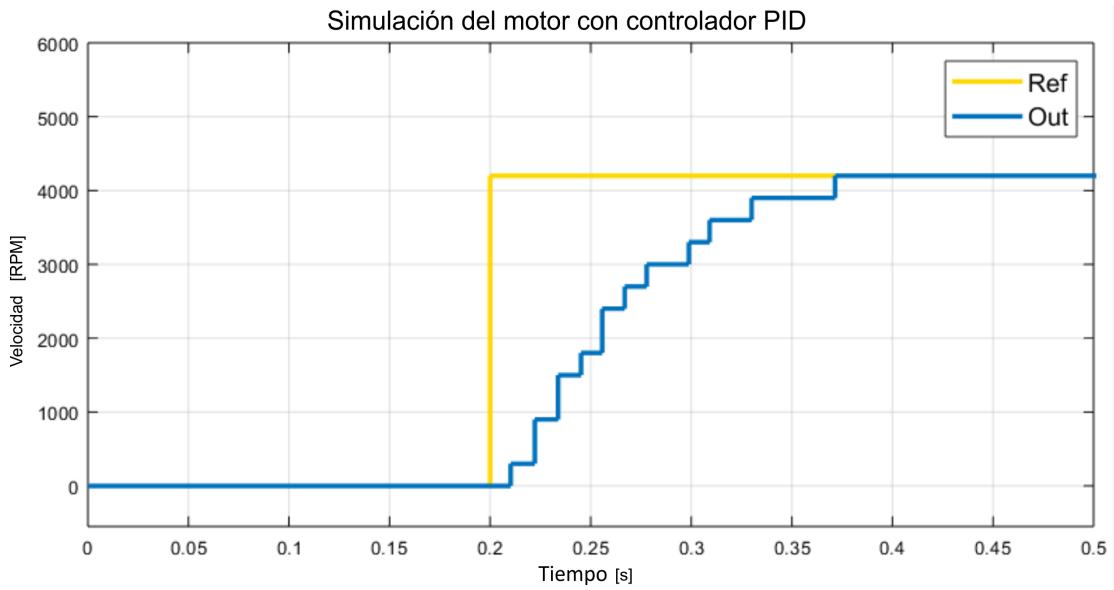
**Figura 17:** Simulink del motor con control PID

## 6.2. Ajuste del Controlador PID



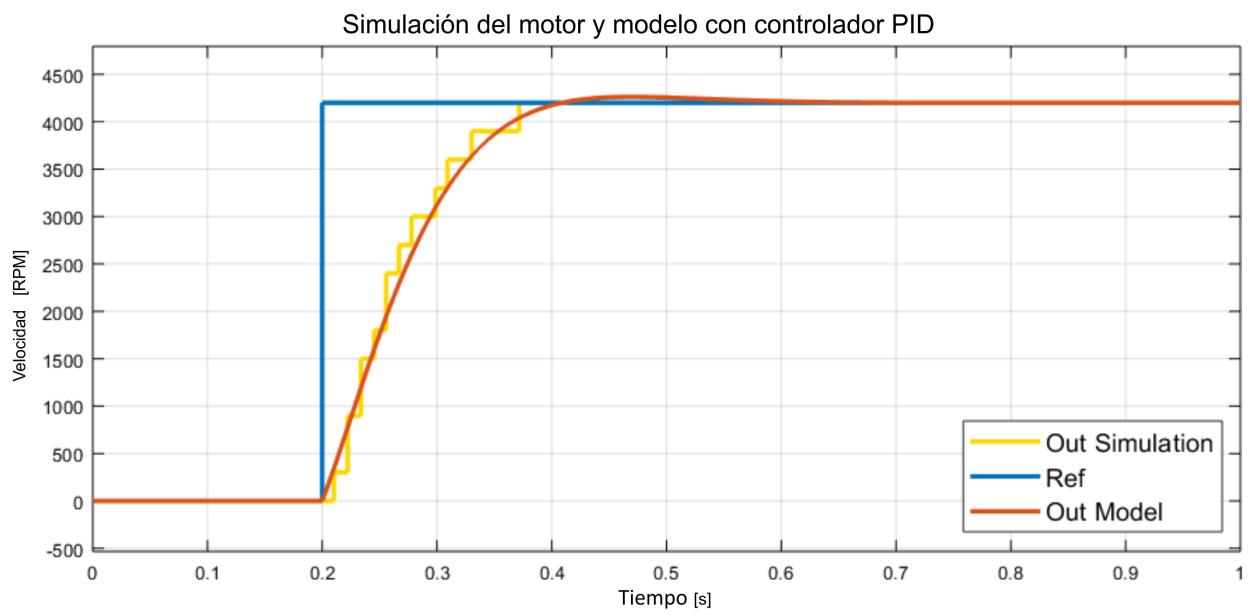
**Figura 18:** Simulación con  $K_i=1$

Tal como se puede observar en la figura 18, la simulación presenta un sobre pico que luego se estabiliza. Para evitar este sobre pico de ajustaron los valores del **PID**, resultando  $K_p=0,001$ ,  $K_d=0,02$ ,  $K_i=0,4$  y  $N=100$ . Luego se realizó una nueva simulación con los nuevos valores, el resultado se puede observar en la figura 19, se puede observar que en este caso, la curva posee el comportamiento deseado.



**Figura 19:** Simulación con  $K_i=0,4$

Por último, se utilizó la misma validación que en el caso anterior, con el hecho de encontrar una transferencia que simule el comportamiento del sistema con el PID. La simulación correspondiente se puede observar en la figura 20.

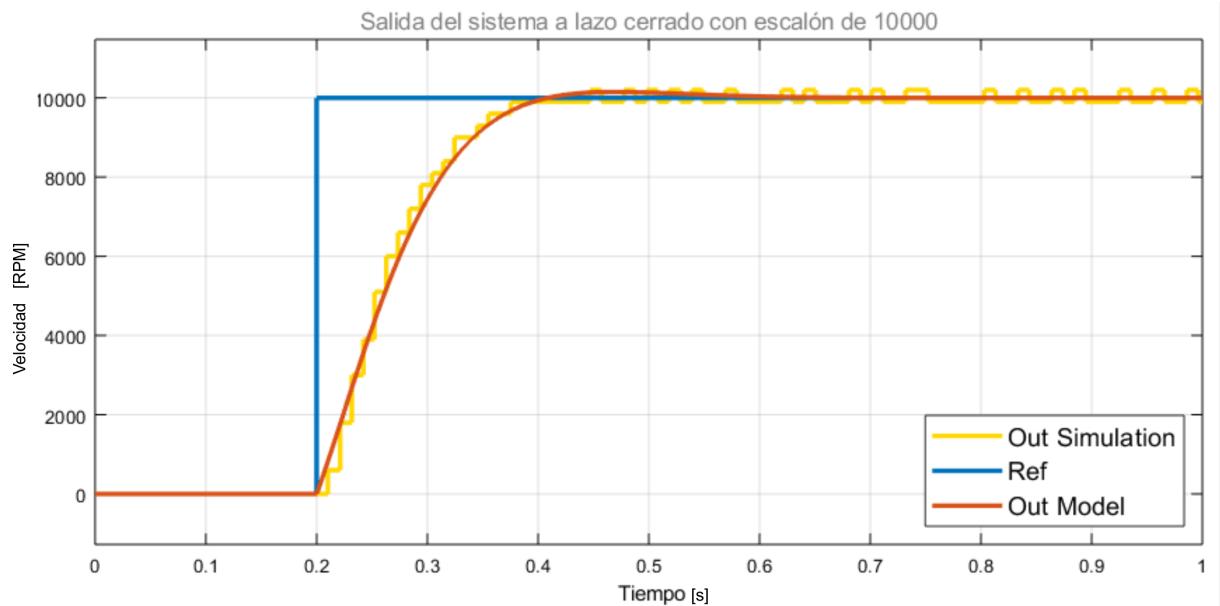


**Figura 20:** Simulación del sistema y su transferencia a lazo cerrado.

## 7. Ensayos en lazo cerrado

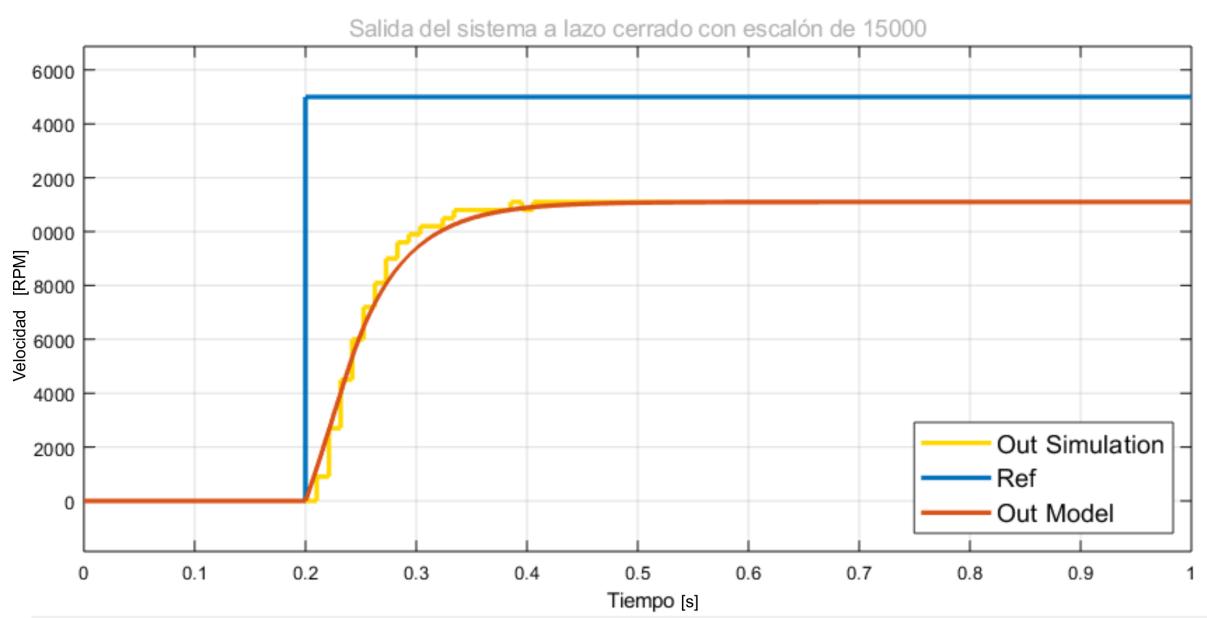
En esta sección se muestran ensayos en lazo cerrado del controlador de velocidad del motor.

- a) Respuesta al escalón de 10000 RPM de amplitud en la entrada de perturbación.



**Figura 21:** Simulación con escalón de 10000.

- b) Respuesta a escalón de 15000 RPM de amplitud en la entrada de perturbación. En este caso se buscó que sature el PID, de modo tal que permita ser observado el funcionamiento del bloque saturador, el cuál limita la salida del bloque del PID a 30 V.



**Figura 22:** Simulación con escalón de 15000.

## 8. Conclusiones

Se concluye que las *S-Functions* son una herramienta útil a la hora de modelar sistemas reales. Con el uso adecuado y en conjunto con *Simulink* es posible simular sistemas y observar su comportamiento al variar diferentes parámetros y excitándolos de diferente manera.

Por otro lado se utilizó la función de un PID realizada para la guía de ejercicios y se comprobó su correcto funcionamiento en una aplicación. Variando los parámetros correspondientes al PID se observó las modificaciones que se realizan sobre el control del sistema.

## A. Código Implementado

### A.1. Ejercicio 1: S-function del tanque

```

1 function [sys ,x0 ,str ,ts ,simStateCompliance] = tanque(t ,x ,u ,flag ,R,hmax,A,hini)
%$FUNTMPL General MATLAB S-Function Template
% With MATLAB S-functions , you can define your own ordinary differential
% equations (ODEs), discrete system equations , and/or just about
% any type of algorithm to be used within a Simulink block diagram.
%
7 % The general form of an MATLAB S-function syntax is:
% [SYS,X0,STR,TS,SIMSTATECOMPLIANCE] = SFUNC(T,X,U,FLAG,P1,...,Pn)
%
9 % What is returned by SFUNC at a given point in time, T, depends on the
11 % value of the FLAG, the current state vector , X, and the current
13 % input vector , U.
%
15 % FLAG      RESULT          DESCRIPTION
17 % _____
19 % 0        [SIZES,X0,STR,TS] Initialization , return system sizes in SYS,
21 %           initial state in X0, state ordering strings
23 %           in STR, and sample times in TS.
25 % 1        DX             Return continuous state derivatives in SYS.
27 % 2        DS             Update discrete states SYS = X(n+1)
29 % 3        Y              Return outputs in SYS.
31 % 4        TNEXT          Return next time hit for variable step sample
33 %           time in SYS.
35 % 5
37 % 9        []             Reserved for future (root finding).
39 %                   Termination , perform any cleanup SYS=[].
%
41 % The state vectors , X and X0 consists of continuous states followed
43 % by discrete states .
%
45 % Optional parameters , P1,...,Pn can be provided to the S-function and
47 % used during any FLAG operation .
%
49 % When SFUNC is called with FLAG = 0 , the following information
51 % should be returned :
%
53 %     SYS(1) = Number of continuous states .
55 %     SYS(2) = Number of discrete states .
57 %     SYS(3) = Number of outputs .
%
59 %     SYS(4) = Number of inputs .
%
61 %             Any of the first four elements in SYS can be specified
63 %             as -1 indicating that they are dynamically sized . The
65 %             actual length for all other flags will be equal to the
67 %             length of the input , U .
%
69 %     SYS(5) = Reserved for root finding . Must be zero .
%
71 %     SYS(6) = Direct feedthrough flag (1=yes , 0=no) . The s-function
73 %             has direct feedthrough if U is used during the FLAG=3
75 %             call . Setting this to 0 is akin to making a promise that
77 %             U will not be used during FLAG=3 . If you break the promise
79 %             then unpredictable results will occur .
%
81 %     SYS(7) = Number of sample times . This is the number of rows in TS .
%
83 %
85 %     X0      = Initial state conditions or [] if no states .
%
87 %     STR     = State ordering strings which is generally specified as [] .
%
```

```

% TS      = An m-by-2 matrix containing the sample time
%           (period , offset) information. Where m = number of sample
%           times. The ordering of the sample times must be:
%
% TS = [0      0,      : Continuous sample time.
%       0      1,      : Continuous, but fixed in minor step
%                         sample time.
%
%           PERIOD OFFSET, : Discrete sample time where
%                         PERIOD > 0 & OFFSET < PERIOD.
%
%           -2      0];    : Variable step discrete sample time
%                         where FLAG=4 is used to get time of
%                         next hit.
%
%
% There can be more than one sample time providing
% they are ordered such that they are monotonically
% increasing. Only the needed sample times should be
% specified in TS. When specifying more than one
% sample time, you must check for sample hits explicitly by
% seeing if
%     abs(round((T-OFFSET)/PERIOD) - (T-OFFSET)/PERIOD)
% is within a specified tolerance, generally 1e-8. This
% tolerance is dependent upon your model's sampling times
% and simulation time.
%
%
% You can also specify that the sample time of the S-function
% is inherited from the driving block. For functions which
% change during minor steps, this is done by
% specifying SYS(7) = 1 and TS = [-1 0]. For functions which
% are held during minor steps, this is done by specifying
% SYS(7) = 1 and TS = [-1 1].
%
%
% SIMSTATECOMPLIANCE = Specifies how to handle this block when saving and
%                     restoring the complete simulation state of the
%                     model. The allowed values are: 'DefaultSimState',
%                     'HasNoSimState' or 'DisallowSimState'. If this value
%                     is not specified, then the block's compliance with
%                     simState feature is set to 'UnknownSimState'.
%
%
% Copyright 1990–2010 The MathWorks, Inc.
%
%
% The following outlines the general structure of an S-function.
%
switch flag ,
%
%
% Initialization %
%
case 0,
    [sys ,x0 ,str ,ts ,simStateCompliance]=mdlInitializeSizes(hmax, hini);
%
%
% Derivatives %
%
case 1,
    sys=mdlDerivatives(t ,x ,u ,R ,hmax ,A ,hini );
%
%
% Update %
%

```

```

119 case 2,
120     sys=mdlUpdate(t,x,u);
121
122 %% %%%%%%
123 % Outputs %
124 %% %%%%%%
125 case 3,
126     sys=mdlOutputs(t,x,u,R,hmax,A,hini);
127
128 %% %%%%%%
129 % GetTimeOfNextVarHit %
130 %% %%%%%%
131 case 4,
132     sys=mdlGetTimeOfNextVarHit(t,x,u,R,hmax,A,hini);
133
134 %% %%%%%%
135 % Terminate %
136 %% %%%%%%
137 case 9,
138     sys=mdlTerminate(t,x,u);
139
140 %% %%%%%%
141 % Unexpected flags %
142 %% %%%%%%
143 otherwise
144     DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));
145
146 end
147
148 %end sfuntmpl
149
150 %
151 %
152 % mdlInitializeSizes
153 % Return the sizes , initial conditions , and sample times for the S-function .
154 %
155 %
156 function [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes(hmax,hini)
157
158 %
159 % call simsizes for a sizes structure , fill it in and convert it to a
160 % sizes array .
161 %
162 % Note that in this example , the values are hard coded . This is not a
163 % recommended practice as the characteristics of the block are typically
164 % defined by the S-function parameters .
165 %
166 sizes = simsizes ;
167
168 sizes.NumContStates = 1;
169 sizes.NumDiscStates = 0;
170 sizes.NumOutputs = 2;
171 sizes.NumInputs = 2;
172 sizes.DirFeedthrough = 1;
173 sizes.NumSampleTimes = 1;    % at least one sample time is needed
174
175 sys = simsizes(sizes);
176
177 %
178 % initialize the initial conditions
179 %

```

```

181 if hini>=hmax
182     x0=hmax; % El tanque llego a su nivel maximo y desbordo.
183 elseif hini<=0
184     x0=0; % Tanque vacio
185 else
186     x0=hini;
187 end
188
189 % str is always an empty matrix
190 %
191 str = [];
192 %
193 % initialize the array of sample times
194 %
195 ts = [0 0];
196
197 % Specify the block simStateCompliance. The allowed values are:
198 %   'UnknownSimState', < The default setting; warn and assume DefaultSimState
199 %   'DefaultSimState', < Same sim state as a built-in block
200 %   'HasNoSimState', < No sim state
201 %   'DisallowSimState' < Error out when saving or restoring the model sim state
202 simStateCompliance = 'UnknownSimState';
203
204 %end mdlInitializeSizes
205 %
206 %
207 %mdlDerivatives
208 % Return the derivatives for the continuous states.
209 %
210 %
211 function sys=mdlDerivatives(t,x,u,R,hmax,A,hini)
212
213 %u=[qi , ho]
214 %y=[h , qo]
215
216 if x>=hmax %tanque lleno
217     h_derivada = (1/A)*(u(1)-(1/R)*sqrt(abs(hmax-u(2)))*sign(hmax-u(2)));
218     %Se usa sign por si el caudal va en un sentido o en el otro
219     if h_derivada >= 0 %se esta llenando de mas
220         sys=0;
221     else
222         sys=h_derivada;
223     end
224 elseif x<=0 % el tanque esta vacio
225     h_derivada = (1/A)*(u(1)-(1/R)*sqrt(abs(0-u(2)))*sign(0-u(2)));
226
227     if h_derivada <= 0 % se esta vaciando de mas
228         sys=0;
229     else
230         sys=h_derivada;
231     end
232 else
233     sys = (1/A)*(u(1)-(1/R)*sqrt(abs(x-u(2)))*sign(x-u(2)));
234 end
235
236 % end mdlDerivatives
237 %
238 %

```

```

241 % mdlUpdate
242 % Handle discrete state updates , sample time hits , and major time step
243 % requirements .
244 %
245 %
246 function sys=mdlUpdate(t,x,u)
247 sys = [];
248 %
249 % end mdlUpdate
250 %
251 %
252 %
253 % mdlOutputs
254 % Return the block outputs .
255 %
256 %
257 function sys=mdlOutputs(t,x,u,R,hmax,A,hini)
258 % u=[qi , ho]
259 % x=[h , qo]
260 %
261 if x(1)>=hmax
262     sys=[hmax (1/R)*sqrt(abs(hmax-u(2)))*sign(hmax-u(2))];
263 elseif x(1)<=0
264     sys=[0 (1/R)*sqrt(abs(0-u(2)))*sign(0-u(2))];
265 else
266     sys=[x(1) (1/R)*sqrt(abs(x(1)-u(2)))*sign(x(1)-u(2))];
267 end
268 %
269 % end mdlOutputs
270 %
271 %
272 %
273 % mdlGetTimeOfNextVarHit
274 % Return the time of the next hit for this block . Note that the result is
275 % absolute time . Note that this function is only used when you specify a
276 % variable discrete-time sample time [-2 0] in the sample time array in
277 % mdlInitializeSizes .
278 %
279 %
280 function sys=mdlGetTimeOfNextVarHit(t,x,u,R,hmax,A,hini)
281 %
282 sampleTime = 1;      % Example , set the next hit to be one second later .
283 sys = t + sampleTime;
284 %
285 % end mdlGetTimeOfNextVarHit
286 %
287 %
288 %
289 % mdlTerminate
290 % Perform any end of simulation tasks .
291 %
292 %
293 function sys=mdlTerminate(t,x,u)
294 sys = [];
295 %
296 % end mdlTerminate

```

tanque.m

## A.2. Ejercicio 2: S-function del motor

```

function [ sys ,x0,str ,ts ,simStateCompliance] = speed(t ,x,u,flag ,Ts,ppv,Tw)
2
% Ts: Tiempo de muestreo
4
% ppv: Pulso por vuelta
% Tw: Tiempo de ventana
6
%$FUNTMPL General MATLAB S-Function Template
8
% With MATLAB S-functions , you can define your own ordinary differential
% equations (ODEs), discrete system equations, and/or just about
10
% any type of algorithm to be used within a Simulink block diagram.
%
12
% The general form of an MATLAB S-function syntax is :
% [SYS,X0,STR,TS,SIMSTATECOMPLIANCE] = SFUNC(T,X,U,FLAG,P1,...,Pn)
%
14
% What is returned by SFUNC at a given point in time, T, depends on the
16
% value of the FLAG, the current state vector , X, and the current
% input vector , U.
%
18
%

| FLAG | RESULT            | DESCRIPTION                                                                                                                    |
|------|-------------------|--------------------------------------------------------------------------------------------------------------------------------|
| 0    | [SIZES,X0,STR,TS] | Initialization , return system sizes in SYS,<br>initial state in X0, state ordering strings<br>in STR, and sample times in TS. |
| 1    | DX                | Return continuous state derivatives in SYS.                                                                                    |
| 2    | DS                | Update discrete states SYS = X(n+1)                                                                                            |
| 3    | Y                 | Return outputs in SYS.                                                                                                         |
| 4    | TNEXT             | Return next time hit for variable step sample<br>time in SYS.                                                                  |
| 5    |                   | Reserved for future (root finding).                                                                                            |
| 9    | []                | Termination , perform any cleanup SYS=[].                                                                                      |


%
32
%
34
% The state vectors , X and X0 consists of continuous states followed
% by discrete states .
%
36
%
38
% Optional parameters , P1,...,Pn can be provided to the S-function and
% used during any FLAG operation .
%
40
%
42
% When SFUNC is called with FLAG = 0 , the following information
% should be returned :
%
44
%

SYS(1) = Number of continuous states .  

%SYS(2) = Number of discrete states .  

%SYS(3) = Number of outputs .  

%SYS(4) = Number of inputs .


%
46
%

Any of the first four elements in SYS can be specified  

%as -1 indicating that they are dynamically sized . The  

%actual length for all other flags will be equal to the  

%length of the input , U .


%
48
%
50
%

SYS(5) = Reserved for root finding . Must be zero .


%
52
%

SYS(6) = Direct feedthrough flag (1=yes , 0=no) . The s-function  

%has direct feedthrough if U is used during the FLAG=3  

%call . Setting this to 0 is akin to making a promise that  

%U will not be used during FLAG=3 . If you break the promise  

%then unpredictable results will occur .


%
54
%
56
%

SYS(7) = Number of sample times . This is the number of rows in TS .


%
58
%
%

X0 = Initial state conditions or [] if no states .


```

```

60 %
61 %
62 % STR = State ordering strings which is generally specified as [].
63 %
64 % TS = An m-by-2 matrix containing the sample time
65 % (period, offset) information. Where m = number of sample
66 % times. The ordering of the sample times must be:
67 %
68 % TS = [0      0,      : Continuous sample time.
69 %           0      1,      : Continuous, but fixed in minor step
70 %                           sample time.
71 %           PERIOD OFFSET, : Discrete sample time where
72 %                           PERIOD > 0 & OFFSET < PERIOD.
73 %           -2      0];    : Variable step discrete sample time
74 %                           where FLAG=4 is used to get time of
75 %                           next hit.
76 %
77 % There can be more than one sample time providing
78 % they are ordered such that they are monotonically
79 % increasing. Only the needed sample times should be
80 % specified in TS. When specifying more than one
81 % sample time, you must check for sample hits explicitly by
82 % seeing if
83 %     abs(round((T-OFFSET)/PERIOD) - (T-OFFSET)/PERIOD)
84 % is within a specified tolerance, generally 1e-8. This
85 % tolerance is dependent upon your model's sampling times
86 % and simulation time.
87 %
88 % You can also specify that the sample time of the S-function
89 % is inherited from the driving block. For functions which
90 % change during minor steps, this is done by
91 % specifying SYS(7) = 1 and TS = [-1 0]. For functions which
92 % are held during minor steps, this is done by specifying
93 % SYS(7) = 1 and TS = [-1 1].
94 %
95 % SIMSTATECOMPLIANCE = Specifies how to handle this block when saving and
96 %                         restoring the complete simulation state of the
97 %                         model. The allowed values are: 'DefaultSimState',
98 %                         'HasNoSimState' or 'DisallowSimState'. If this value
99 %                         is not specified, then the block's compliance with
100 %                         simState feature is set to 'UnknownSimState'.
101 %
102 % Copyright 1990–2010 The MathWorks, Inc.
103 %
104 % The following outlines the general structure of an S-function.
105 %
106 switch flag,
107
108 %%%
109 %% Initialization %
110 %%%
111 case 0,
112     [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes(Ts);
113 %%%
114 %%%
115 %% Derivatives %
116 %%%
117 case 1,
118     sys=mdlDerivatives(t,x,u);
119

```

```

122    %%%
123    % Update %
124    %%%
125    case 2,
126        sys=mdlUpdate(t,x,u,ppv,Tw);
127
128    %%%
129    % Outputs %
130    %%%
131    case 3,
132        sys=mdlOutputs(t,x,u,ppv,Tw);
133
134    %%%
135    % GetTimeOfNextVarHit %
136    %%%
137    case 4,
138        sys=mdlGetTimeOfNextVarHit(t,x,u,Ts);
139
140    %%%
141    % Terminate %
142    %%%
143    case 9,
144        sys=mdlTerminate(t,x,u);
145
146    %%%
147    % Unexpected flags %
148    %%%
149    otherwise
150        DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));
151
152    end
153
154    % end sfuntmpl
155
156    %
157    %=====
158    % mdlInitializeSizes
159    % Return the sizes, initial conditions, and sample times for the S-function.
160    %=====
161
162    %
163    % call simsizes for a sizes structure, fill it in and convert it to a
164    % sizes array.
165    %
166    % Note that in this example, the values are hard coded. This is not a
167    % recommended practice as the characteristics of the block are typically
168    % defined by the S-function parameters.
169    %
170    sizes = simsizes;
171
172    sizes.NumContStates = 0;
173    sizes.NumDiscStates = 4;
174    sizes.NumOutputs = 1;
175    sizes.NumInputs = 1;
176    sizes.DirFeedthrough = 1;
177    sizes.NumSampleTimes = 1;    % at least one sample time is needed
178
179    sys = simsizes(sizes);

```

```

182 %
183 % initialize the initial conditions
184 %
185 x0 = [1 0 0 0]; % Inicializo el estado anterior en 1,
186 % para que no entre al if u > x(1),
187 % que se encuentra en mdlUpdate()
188 %
189 % str is always an empty matrix
190 %
191 str = [];
192 %
193 % initialize the array of sample times
194 %
195 ts = [Ts 0];
196 %
197 % Specify the block simStateCompliance. The allowed values are:
198 % 'UnknownSimState', < The default setting; warn and assume DefaultSimState
199 % 'DefaultSimState', < Same sim state as a built-in block
200 % 'HasNoSimState', < No sim state
201 % 'DisallowSimState' < Error out when saving or restoring the model sim state
202 simStateCompliance = 'UnknownSimState';
203 %
204 %end mdlInitializeSizes
205 %
206 %
207 %mdlDerivatives
208 % Return the derivatives for the continuous states.
209 %
210 %
211 function sys=mdlDerivatives(t,x,u)
212 %
213 sys = [];
214 %
215 %end mdlDerivatives
216 %
217 %
218 %mdlUpdate
219 % Handle discrete state updates, sample time hits, and major time step
220 % requirements.
221 %
222 %
223 function sys=mdlUpdate(t,x,u,ppv,Tw)
224 %
225 %x = [estado anterior , contador , tiempo inicial de ventana , velocidad]
226 %u = [tension de entrada]
227 %
228 if x(2) == 0
229     x(3) = t;
230 end
231 %
232 if u > x(1)
233     x(2) = x(2)+1; %Se encontro un pulso, por lo que se aumenta el contador
234 end
235 %
236 if (t-x(3)) > Tw %Se paso la ventana de tiempo
237     %Defino la velocidad a partir de la cantidad de pulsos contados:
238     x(4) = (60*x(2))/(ppv*Tw); %Escribo a la velocidad en RPM
239 end
240 
```

```

244 x(2) = 0; % Inicializo el contador en 0 para la proxima ventana de tiempo2
end

246 x(1)=u;

248 sys = x;

250 % end mdlUpdate

252 %
%=====mdlOutputs=====
254 % mdlOutputs
% Return the block outputs.
%=====mdlOutputs=====

258 function sys=mdlOutputs(t,x,u,ppv,Tw)
% x = [estado anterior , contador , tiempo inicial de ventana , velocidad]
260 sys = [x(4)];
262 %end mdlOutputs

264 %
%=====mdlGetTimeOfNextVarHit=====
266 % mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block. Note that the result is
% absolute time. Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
%=====mdlGetTimeOfNextVarHit=====

274 function sys=mdlGetTimeOfNextVarHit(t,x,u,Ts)

276 sampleTime = 1; % Example, set the next hit to be one second later.
sys = t + sampleTime;
278 %end mdlGetTimeOfNextVarHit

280 %
%=====mdlTerminate=====
282 % mdlTerminate
% Perform any end of simulation tasks.
%=====mdlTerminate=====

286 function sys=mdlTerminate(t,x,u)
288 sys = [];
290 %end mdlTerminate

```

dcmotor\_speed.m

### A.3. Ejercicio 3: S-function del PID

```

1 function [ sys ,x0 ,str ,ts ,simStateCompliance] = pid_sfunc(t ,x,u,flag ,Kp,Kd,Ki ,h,N)
2 %$FUNIMPL General MATLAB S-Function Template
3 % With MATLAB S-functions , you can define your own ordinary differential
4 % equations (ODEs) , discrete system equations , and/or just about
5 % any type of algorithm to be used within a Simulink block diagram.
6 %
7 % The general form of an MATLAB S-function syntax is:
8 % [SYS,X0,STR,TS,SIMSTATECOMPLIANCE] = SFUNC(T,X,U,FLAG,P1,...,Pn)
9 %
10 % What is returned by SFUNC at a given point in time , T, depends on the
11 % value of the FLAG, the current state vector , X, and the current
12 % input vector , U.
13 %
14 %

| FLAG | RESULT             | DESCRIPTION                                                                                                                        |
|------|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| 0    | [SIZES ,X0,STR,TS] | Initialization , return system sizes in SYS, 27 % initial state in X0, state ordering strings 28 % in STR, and sample times in TS. |
| 1    | DX                 | Return continuous state derivatives in SYS.                                                                                        |
| 2    | DS                 | Update discrete states SYS = X(n+1)                                                                                                |
| 3    | Y                  | Return outputs in SYS.                                                                                                             |
| 4    | TNEXT              | Return next time hit for variable step sample 49 % time in SYS.                                                                    |
| 5    |                    | Reserved for future (root finding).                                                                                                |
| 9    | []                 | Termination , perform any cleanup SYS=[].                                                                                          |


63 %
64 %
65 % The state vectors , X and X0 consists of continuous states followed
66 % by discrete states .
67 %
68 % Optional parameters , P1,...,Pn can be provided to the S-function and
69 % used during any FLAG operation .
70 %
71 % When SFUNC is called with FLAG = 0 , the following information
72 % should be returned :
73 %
74 % SYS(1) = Number of continuous states .
75 % SYS(2) = Number of discrete states .
76 % SYS(3) = Number of outputs .
77 % SYS(4) = Number of inputs .
78 %
79 % Any of the first four elements in SYS can be specified
80 % as -1 indicating that they are dynamically sized . The
81 % actual length for all other flags will be equal to the
82 % length of the input , U .
83 %
84 % SYS(5) = Reserved for root finding . Must be zero .
85 % SYS(6) = Direct feedthrough flag (1=yes , 0=no) . The s-function
86 % has direct feedthrough if U is used during the FLAG=3
87 % call . Setting this to 0 is akin to making a promise that
88 % U will not be used during FLAG=3 . If you break the promise
89 % then unpredictable results will occur .
90 %
91 % SYS(7) = Number of sample times . This is the number of rows in TS .
92 %
93 %
94 % X0 = Initial state conditions or [] if no states .
95 %
96 % STR = State ordering strings which is generally specified as [] .
97 %
98 % TS = An m-by-2 matrix containing the sample time
99 % (period , offset ) information . Where m = number of sample

```

```

%
% times. The ordering of the sample times must be:
61 %
% TS = [0      0,      : Continuous sample time.
63 %      0      1,      : Continuous, but fixed in minor step
%      sample time.
65 %      PERIOD OFFSET, : Discrete sample time where
%      PERIOD > 0 & OFFSET < PERIOD.
67 %      -2      0];    : Variable step discrete sample time
%      where FLAG=4 is used to get time of
69 %      next hit.
%
71 %
% There can be more than one sample time providing
% they are ordered such that they are monotonically
73 % increasing. Only the needed sample times should be
% specified in TS. When specifying more than one
75 % sample time, you must check for sample hits explicitly by
% seeing if
77 %      abs(round((T-OFFSET)/PERIOD) - (T-OFFSET)/PERIOD)
%      is within a specified tolerance, generally 1e-8. This
79 %      tolerance is dependent upon your model's sampling times
%      and simulation time.
81 %

%
% You can also specify that the sample time of the S-function
% is inherited from the driving block. For functions which
% change during minor steps, this is done by
% specifying SYS(7) = 1 and TS = [-1 0]. For functions which
% are held during minor steps, this is done by specifying
% SYS(7) = 1 and TS = [-1 1].
%
89 % SIMSTATECOMPLIANCE = Specifies how to handle this block when saving and
%      restoring the complete simulation state of the
91 %      model. The allowed values are: 'DefaultSimState',
%      'HasNoSimState' or 'DisallowSimState'. If this value
93 %      is not specified, then the block's compliance with
%      simState feature is set to 'UnknownSimState'.
95

97 % Copyright 1990–2010 The MathWorks, Inc.

99 %
% The following outlines the general structure of an S-function.
101 %
switch flag ,
103
105 % Initialization %
106 %%%
107 case 0,
108     [sys ,x0 ,str ,ts ,simStateCompliance]=mdlInitializeSizes ;
109
110 %%%
111 % Derivatives %
112 %%%
113 case 1,
114     sys=mdlDerivatives (t ,x ,u ,Kp ,Kd ,Ki ,h ,N) ;
115
116 %%%
117 % Update %
118 %%%
119 case 2,
120     sys=mdlUpdate (t ,x ,u ,Kp ,Kd ,Ki ,h ,N) ;

```

```

121    %%%
122    % Outputs %
123    %%%
124    case 3,
125        sys=mdlOutputs(t,x,u,Kp,Kd,Ki,h,N);
126
127    %%%
128    % GetTimeOfNextVarHit %
129    %%%
130    case 4,
131        sys=mdlGetTimeOfNextVarHit(t,x,u);
132
133    %%%
134    % Terminate %
135    %%%
136    case 9,
137        sys=mdlTerminate(t,x,u);
138
139    %%%
140    % Unexpected flags %
141    %%%
142    otherwise
143        DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));
144
145    end
146
147    % end sfuntmpl
148
149    %
150    %
151    % mdlInitializeSizes
152    % Return the sizes, initial conditions, and sample times for the S-function.
153    %
154    %
155    function [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes
156
157    %
158    % call simsizes for a sizes structure, fill it in and convert it to a
159    % sizes array.
160    %
161    % Note that in this example, the values are hard coded. This is not a
162    % recommended practice as the characteristics of the block are typically
163    % defined by the S-function parameters.
164    %
165    sizes = simsizes;
166
167    sizes.NumContStates = 0;
168    sizes.NumDiscStates = 3;
169    sizes.NumOutputs = 1;
170    sizes.NumInputs = 2;
171    sizes.DirFeedthrough = 1;
172    sizes.NumSampleTimes = 1;    % at least one sample time is needed
173
174    sys = simsizes(sizes);
175
176    %
177    % initialize the initial conditions
178    %
179    x0 = [0 0 0];
180

```

```

183 %
184 % str is always an empty matrix
185 %
186 str = [];
187 %
188 % initialize the array of sample times
189 %
190 ts = [0 0];
191 %
192 % Specify the block simStateCompliance. The allowed values are:
193 %   'UnknownSimState', < The default setting; warn and assume DefaultSimState
194 %   'DefaultSimState', < Same sim state as a built-in block
195 %   'HasNoSimState', < No sim state
196 %   'DisallowSimState' < Error out when saving or restoring the model sim state
197 simStateCompliance = 'UnknownSimState';
198 %
199 %end mdlInitializeSizes
200 %
201 %
202 %mdlDerivatives
203 % Return the derivatives for the continuous states.
204 %
205 %
206 function sys=mdlDerivatives(t,x,u,Kp,Kd,h,N)
207 %
208 sys = [];
209 %
210 %end mdlDerivatives
211 %
212 %
213 %mdlUpdate
214 % Handle discrete state updates, sample time hits, and major time step
215 % requirements.
216 %
217 %
218 function sys=mdlUpdate(t,x,u,Kp,Kd,Ki,h,N)
219 %
220 gamma=Kd/N;
221 A=[1 0 0; 0 gamma/(gamma+h) (Kp*Kd)/(gamma+h); 0 0 0];
222 B=[h*Kp*Ki -h*Kp*Ki; 0 -Kp*Kd/(gamma+h); 0 1];
223 %
224 sys = [A*x+B*u];
225 %
226 %end mdlUpdate
227 %
228 %
229 %mdlOutputs
230 % Return the block outputs.
231 %
232 %
233 function sys=mdlOutputs(t,x,u,Kp,Kd,Ki,h,N)
234 %
235 gamma=Kd/N;
236 %
237 C=[1 gamma/(gamma+h) Kp*Kd/(gamma+h)];
238 D=[Kp (-Kp-(Kp*Kd/(gamma+h)))];
239 
```

```

243 sys = [C*x+D*u];
245 % end mdlOutputs
247 %
248 %
249 % mdlGetTimeOfNextVarHit
250 % Return the time of the next hit for this block. Note that the result is
251 % absolute time. Note that this function is only used when you specify a
252 % variable discrete-time sample time [-2 0] in the sample time array in
253 % mdlInitializeSizes.
254 %
255 %
256 function sys=mdlGetTimeOfNextVarHit(t,x,u)
257
258 sampleTime = 1;      % Example, set the next hit to be one second later.
259 sys = t + sampleTime;
260
261 % end mdlGetTimeOfNextVarHit
262 %
263 %
264 % mdlTerminate
265 % Perform any end of simulation tasks.
266 %
267 %
268 function sys=mdlTerminate(t,x,u)
269
270 sys = [];
271
272 % end mdlTerminate

```

pid\_sfunc.m