Ingeniería Informática

Sistemas de Control por Computador (SCC)

Práctica 2

Cálculo del error en estado estable de un sistema de control

Curso 2008-09

18/11/2008

Considere el diagrama de bloques de un sistema de control mostrado en la siguiente figura.

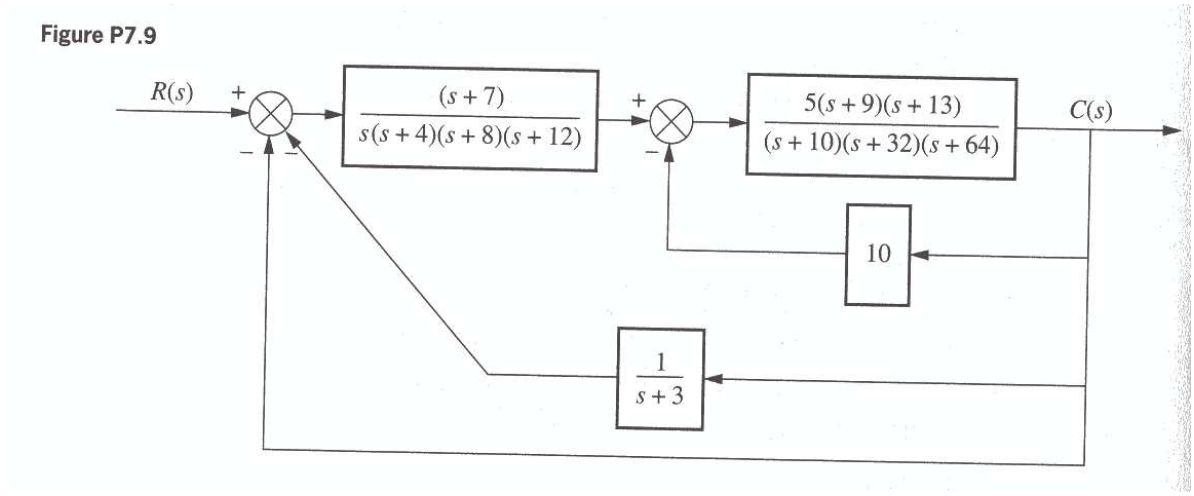


Figura 1: Diagrama de bloques de un sistema de control

Utilice MATLAB para encontrar

1. El tipo del sistema
2. Las constantes K_p, K_v, K_a.
3. El error en estado estable para las entradas: $30u(t)$, $30tu(t)$ y $30t^2u(t)$.

Para resolver esta práctica se incluyen a continuación un ejemplo de simplificación de diagramas de bloques usando MATLAB y un ejemplo del cálculo del error en estado estable de un sistema de control también con MATLAB.

1. Ejemplo de simplificación de diagrama de bloques

La figura 2 muestra un vehículo UFSS (*Unmanned Free-Swimming Submersible*). La profundidad de este vehículo se controla moviendo una superficie elevadora durante el movimiento de avance. La desviación de esta superficie provoca una fuerza vertical que hace que el vehículo rote sobre su eje de cabezada y que por tanto se sumerja o ascienda a la superficie. En la figura 3 se muestra el diagrama de bloques del sistema de control de la superficie elevadora.

MATLAB puede emplearse para reducir el diagrama de bloques de la figura 3. Para determinar los coeficientes de los polinomios de una función de transferencia se pueden utilizar las funciones *poly* y *conv*. La función *poly* tiene como argumento un vector con las raíces del polinomio y produce como salida un vector con los coeficientes del polinomio. Por ejemplo, en un polinomio con raíces

$$r = [-3 \ -2 \ -1]$$

la instrucción

$$q = poly(r)$$

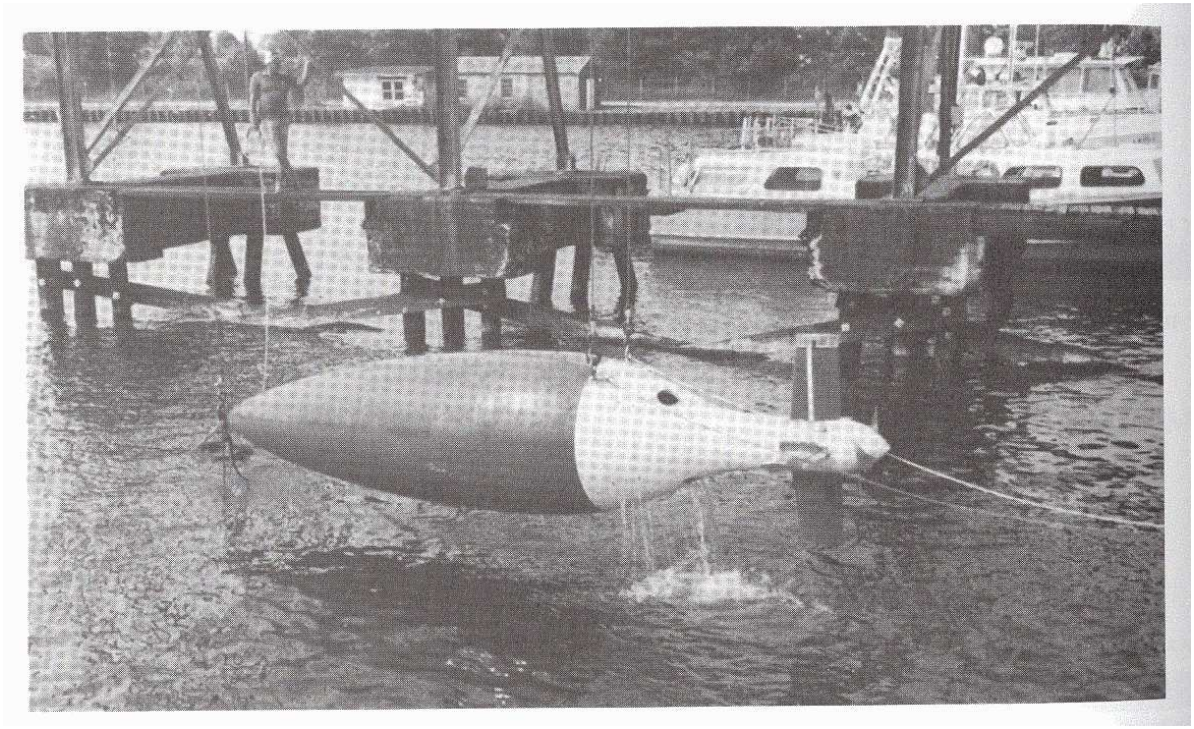


Figura 2: Vehículo UFSS (Unmanned Free-Swimming Submersible)

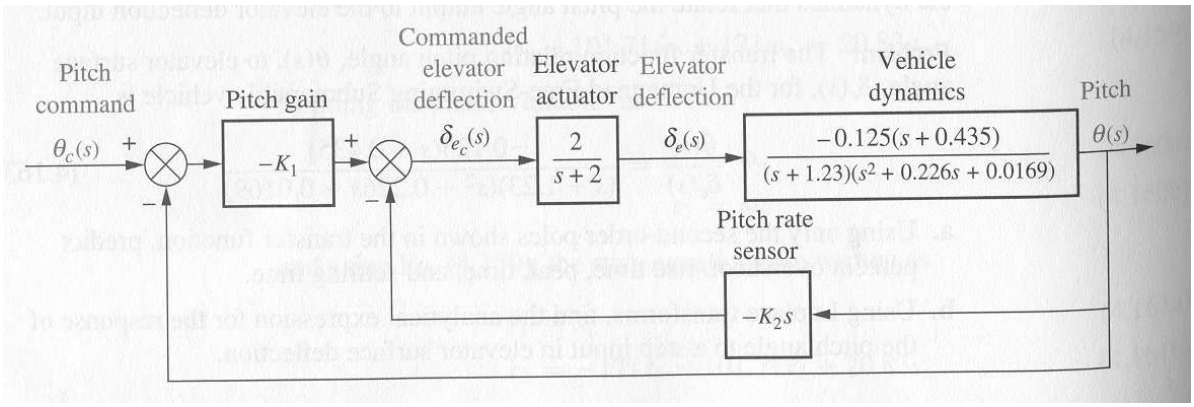


Figura 3: Diagrama de bloques del sistema de control de profundidad del vehículo UFSS

da lugar al vector de coeficientes

$$q = [1\ 6\ 11\ 6]$$

Esto siginifica que

$$s^3 + 6s^2 + 11s + 6 = (s+3)(s+2)(s+1)$$

La función *conv* realiza el producto de dos polinomios. Por ejemplo, sean dos polinomios

$$\begin{aligned} a &= s^2 - 20{,}6 \\ b &= s^2 + 19{,}6s + 151{,}2 \end{aligned}$$

Ahora definimos los vectores

$$\begin{aligned} a &= [1\ 0\ -20{,}6] \\ b &= [1\ 19{,}6\ 151{,}2] \end{aligned}$$

Introduciend la instrucción

$$c = conv(a, b)$$

se obtiene el resultado

$$c = [1\ 19{,}6\ 130{,}6\ -403{,}8\ 3114{,}7]$$

que es la representación en MATLAB del polinomio

$$s^4 + 19{,}6s^3 + 130{,}6s^2 - 403{,}8s + 3114{,}7$$

Los sistemas pueden simplificarse utilizando los siguientes comandos cuyos argumentos son objetos LTI (*Linear Time Invariant systems*): `series(G1,G2)`, que permite simplificar la interconexión en serie de $G_1(s)$ y $G_2(s)$; `parallel(G1,G2)`, que permite simplificar la interconexión en paralelo de $G_1(s)$ y $G_2(s)$; y `feedback(G,H,sign)` que permite simplificar un sistema realimentado con $G(s)$ en el camino directo, $H(s)$ en el camino realimentado y con el parámetro `sign` igual a -1 si la realimentación es negativa o igual a $+1$ si la realimentación es positiva. Un objeto LTI se crea a partir de los coeficientes de los polinomios numerador y denominador de la función de transferencia[1].

Para calcular el error en estado estable, MATLAB dispone de la función *dcgain* que tiene como argumento el objeto LTI que representa a una función de transferencia y produce como salida

$$\lim_{s\to 0}\ G(s)$$

En algunas ocasiones, hay que simplificar una función de transferencia eliminando raíces comunes en el numerador y en el denominador. Esto puede hacerse automáticamente en MATLAB utilizando la instrucción *minreal* que toma como argumento un objeto LTI y produce como salida otro objeto LTI simplificado.

A continuación se muestra un ejemplo de programa en MATLAB que permite simplificar el diagrama de bloques del sistema de control de la figura 3.

[1]Introducir *help tf* para consultar la sintaxis de la instrucción *tf*

```
'(ch5p1) UFSS Pitch Control System' % Display label.
'Solution via Series, Parallel, & Feedback Commands'
                                    % Display label.
numg1=[-1];                         % Define numerator of G1(s).
deng1=[1];                          % Define denominator of G1(s).
numg2=[0 2];                        % Define numerator of G2(s).
deng2=[1 2];                        % Define denominator of G2(s).
numg3=-0.125*[1 0.435];             % Define numerator of G3(s).
deng3=conv([1 1.23],[1 0.226 0.0169]);
                                    % Define denominator of G3(s).
numh1=[-1 0];                       % Define numerator of H1(s).
denh1=[0 1];                        % Define denominator of H1(s).
G1=tf(numg1,deng1);                 % Create LTI transfer function,
                                    % G1(s).
G2=tf(numg2,deng2);                 % Create LTI transfer function,
                                    % G2(s).
G3=tf(numg3,deng3);                 % Create LTI transfer function,
                                    % G3(s).
H1=tf(numh1,denh1);                 % Create LTI transfer function,
                                    % H1(s).
G4=series(G2,G3);                   % Calculate product of elevator
                                    % and vehicle dynamics.
G5=feedback(G4,H1);                 % Calculate closed-loop transfer
                                    % function of inner loop.
Ge=series(G1,G5);                   % Multiply inner-loop transfer
                                    % function and pitch gain.
'T(s) via Series, Parallel, & Feedback Commands'
                                    % Display label.
T=feedback(Ge,1)                    % Find closed-loop transfer
                                    % function.
```

Figura 4: Programa en MATLAB para la simplificación del diagrama de bloques de la figura 3.

2. Ejemplo de cálculo del error en estado estable

Para calcular el error en estado estable, MATLAB dispone de la función *dcgain* que tiene como argumento el objeto LTI que representa a una función de transferencia y produce como salida

$$\lim_{s\to 0} G(s)$$

En algunas ocasiones, hay que simplificar una función de transferencia eliminando raíces comunes en el numerador y en el denominador. Esto puede hacerse automáticamente en MATLAB utilizando la instrucción *minreal* que toma como argumento un objeto LTI y produce como salida otro objeto LTI simplificado.

A continuación se muestra un ejemplo de programa en MATLAB para calcular el error en estado estable de un sistema de control ante diversas entradas.

Steady-state error via static error constants

Problem For each system of Figure 7.7, evaluate the static error constants and find the expected error for the standard step, ramp, and parabolic inputs.

Solution First, verify that all closed-loop systems shown are indeed stable. For this example we leave out the details. Next, for Figure 7.7(*a*),

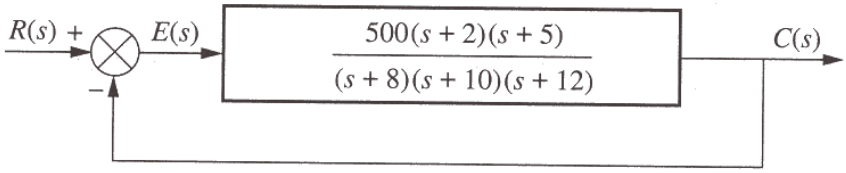
$$K_p = \lim_{s\to 0} G(s) = \frac{500 \times 2 \times 5}{8 \times 10 \times 12} = 5.208 \tag{7.36}$$

$$K_v = \lim_{s\to 0} sG(s) = 0 \tag{7.37}$$

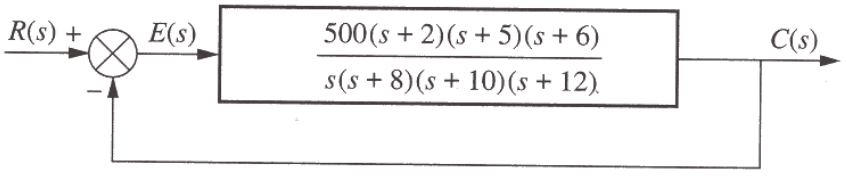
$$K_a = \lim_{s\to 0} s^2 G(s) = 0 \tag{7.38}$$

Thus, for a step input,

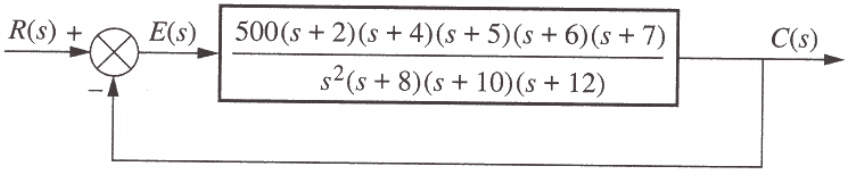
$$e(\infty) = \frac{1}{1 + K_p} = 0.161 \tag{7.39}$$



(*a*)



(*b*)



(*c*)

For a ramp input,

$$e(\infty) = \frac{1}{K_v} = \infty$$

For a parabolic input,

$$e(\infty) = \frac{1}{K_a} = \infty$$

Now, for Figure 7.7(*b*),

$$K_p = \lim_{s \to 0} G(s) = \infty$$

$$K_v = \lim_{s \to 0} sG(s) = \frac{500 \times 2 \times 5 \times 6}{8 \times 10 \times 12} = 31.25$$

and

$$K_a = \lim_{s \to 0} s^2 G(s) = 0$$

Thus, for a step input,

$$e(\infty) = \frac{1}{1 + K_p} = 0$$

For a ramp input,

$$e(\infty) = \frac{1}{K_v} = \frac{1}{31.25} = 0.032$$

For a parabolic input,

$$e(\infty) = \frac{1}{K_a} = \infty$$

Finally, for Figure 7.7(c),

$$K_p = \lim_{s \to 0} G(s) = \infty$$

$$K_v = \lim_{s \to 0} sG(s) = \infty$$

and

$$K_a = \lim_{s \to 0} s^2 G(s) = \frac{500 \times 2 \times 4 \times 5 \times 6 \times 7}{8 \times 10 \times 12} = 875$$

Thus, for a step input,

$$e(\infty) = \frac{1}{1 + K_p} = 0$$

For a ramp input,

$$e(\infty) = \frac{1}{K_v} = 0$$

For a parabolic input,

$$e(\infty) = \frac{1}{K_a} = \frac{1}{875} = 1.14 \times 10^{-3}$$

Chapter 7: Steady-State Errors

ch7p1 (Example 7.4, sys. b) Static error constants are found using $\lim s^n G(s)$ as $s \to 0$. Once the static error constant is found, we can evaluate the steady-state error. To evaluate the static error constant we can use the command `dcgain(G)`, which evaluates $G(s)$ at $s = 0$. Let us look at Example 7.4, system **b**, in the text.

```
'(ch7p1) Example 7.4, sys. b'        % Display label.
numg=500*poly([-2 -5 -6]);           % Define numerator of G(s).
deng=poly([0 -8 -10 -12]);           % Define denominator of G(s).
G=tf(numg,deng);                     % Form G(s)
'Check Stability'                    % Display label.
T=feedback(G,1);                     % Form T(s).
poles=pole(T)                        % Display closed-loop poles.
'Step Input'                         % Display label.
Kp=dcgain(G)                         % Evaluate Kp=numg/deng for s=0.
ess=1/(1+Kp)                         % Evaluate ess for step input.
'Ramp Input'                         % Display label.
numsg=conv([1 0],numg);              % Define numerator of sG(s).
densg=poly([0 -8 -10 -12]);          % Define denominator of sG(s).
sG=tf(numsg,densg);                  % Create sG(s).

sG=minreal(sG);                      % Cancel common 's' in
                                     % numerator(numsg) and
                                     % denominator(densg).
Kv=dcgain(sG)                        % Evaluate Kv=sG(s) for s=0.
ess=1/Kv                             % Evaluate steady-state error for
                                     % ramp input.
'Parabolic Input'                    % Display label.
nums2g=conv([1 0 0],numg);           % Define numerator of s^2G(s).
dens2g=poly([0 -8 -10 -12]);         % Define denominator of s^2G(s).
s2G=tf(nums2g,dens2g);               % Create s^2G(s).
s2G=minreal(s2G);                    % Cancel common 's' in
                                     % numerator(nums2g) and
                                     % denominator(dens2g).
Ka=dcgain(s2G)                       % Evaluate Ka=s^2G(s) for s=0.
ess=1/Ka                             % Evaluate steady-state error for
                                     % parabolic input.
pause
```