Escuela Universitaria
Politécnica - La Almunia
Centro adscrito
Universidad Zaragoza

ENERO/2023

REGULACIÓN Y CONTROL AUTOMÁTICO

REGULADORES

PROFESOR **Dr. Diego Antolín Cañada**
AUTOR NAOUFAL EL RHAZZALI

UNIVERSIDAD DE ZARAGOZA

CONTENIDO

FIGURAS

TABLAS

ECUACIONES

OBJETO

Diseño de reguladores continuos mediante la herramienta **Tunner de Matlab**, para posteriormente discretizarlo e implementarlos en C/C++. Así mismo, con la versión **discreta del Tunner de Matlab**, diseñar reguladores discretos e implementarlos en C/C++. Se apoyará el desarrollo de este en el software de simulación: **Simulink**

APARTADO I

En este apartado, empleando el **PID tunner** de Matlab (tunner), se diseñarán reguladores (P, PI, PD o PID) que satisfagan las especificaciones solicitadas en cada caso.

A	B	C	D	E	F
7	8	7	2	4	6

Tabla 1 – Decodificación de NIA

Caso 1- Continuo

Se pide un regulador para el siguiente sistema que cumpla: Error de posición cero, sobreoscilación menor al 30% y tiempo de respuesta menor a 0.3".

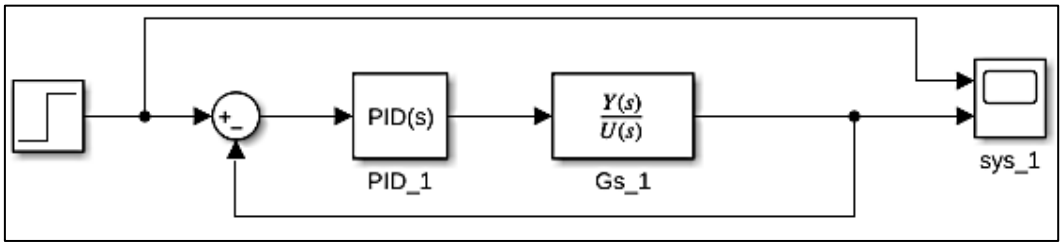

Figura 1 – Diagrama de bloques del sistema – caso 1

[Ec. 1] – Función de transferencia continua general – caso 1

$$G_{s_1} = \frac{A\left(s + \frac{B}{15}\right)}{s + \frac{C}{7}}$$

[Ec. 2] – Función de transferencia continua particular – caso 1

$$G_{s_1} = \frac{7s + 3.733}{s + 1}$$

En el Caso 1, la función de transferencia continua (Fts) de la planta **[Ec. 1]**, es de 1º orden. Por lo tanto, por la propia naturaleza de los sistemas de primer orden, se cumple que la sobreoscilación es nula. Por otra parte, para conseguir un error de posición nulo, se ha de disponer de un integrador (algún polo en cero), hecho que no se observa en la planta.

Por ello, la primera estrategia de diseño sería probar como un regulador integrador (I), así se asegura el error de posición nulo. Así, lo único que quedaría por conseguir es ajustar la ganancia del regulador hasta alcanzar un tiempo de asentamiento de 0.3".

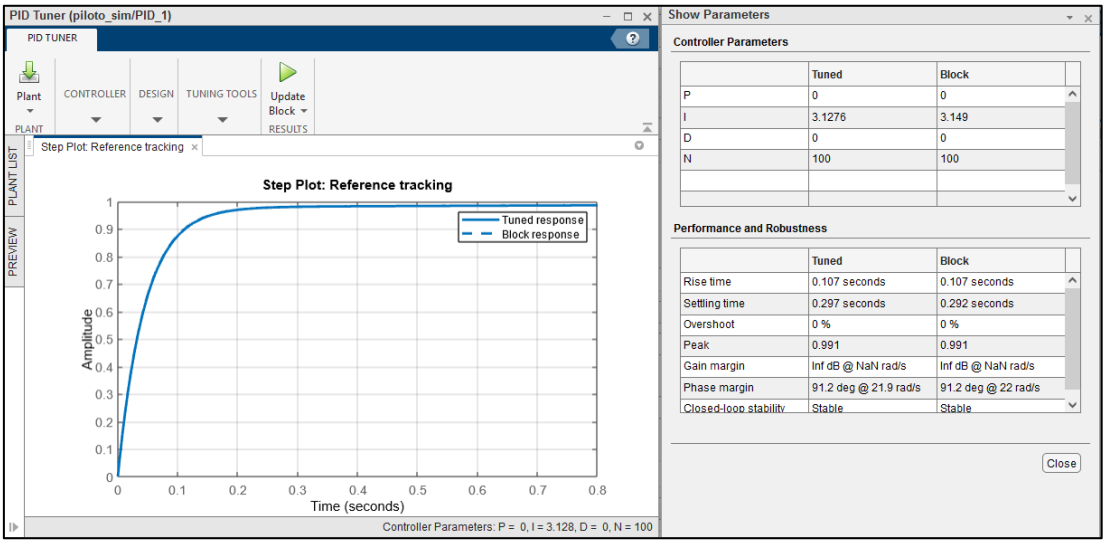

Figura 2 – Ajuste del regulador – caso 1

[Ec. 3] – Función de transferencia continua del Regulador – caso 1

$$R_{s_1} = \frac{3.128}{s}$$

Tras conseguir el regulador **[Ec. 3]**, y así, el ajuste del sistema con el Tunner, en la tabla de **Show Parameters** (Figura 2), se puede ver que se ha conseguido el tiempo de asentamiento requerido.

Evaluación del sistema

En este caso, se procede a evaluar el comportamiento del sistema en el estado temporal (sobreoscilación y tiempo de asentamiento), y en el estado estacionario (errores).

Excitamos el sistema con un escalón unitario, mediante la función **setp(…)** de Matlab.

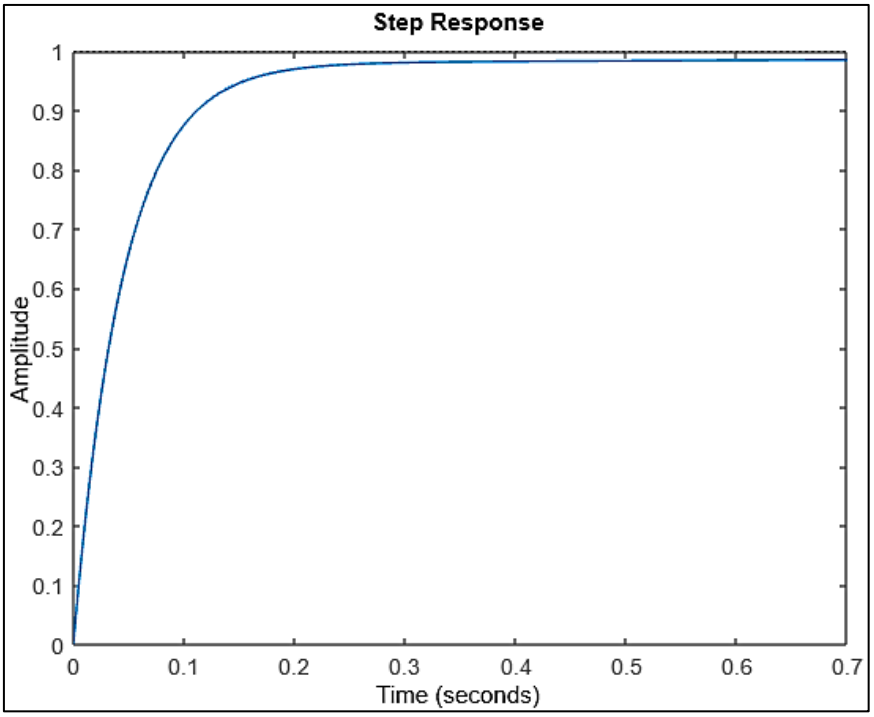

Figura 3 – Respuesta al escalón unitario – caso 1

En la Figura 3 se puede observar que se cumple que la sobreoscilación es nula. Para comprobar que el tiempo de asentamiento sea menor que el 0,3". Entendiendo que dicho tiempo se refiere a que la salida del sistema ha alcanzado el 98% del valor de entrada, entrando en las bandas del ±0.02%. Se procede para ello, con **Simulink**.

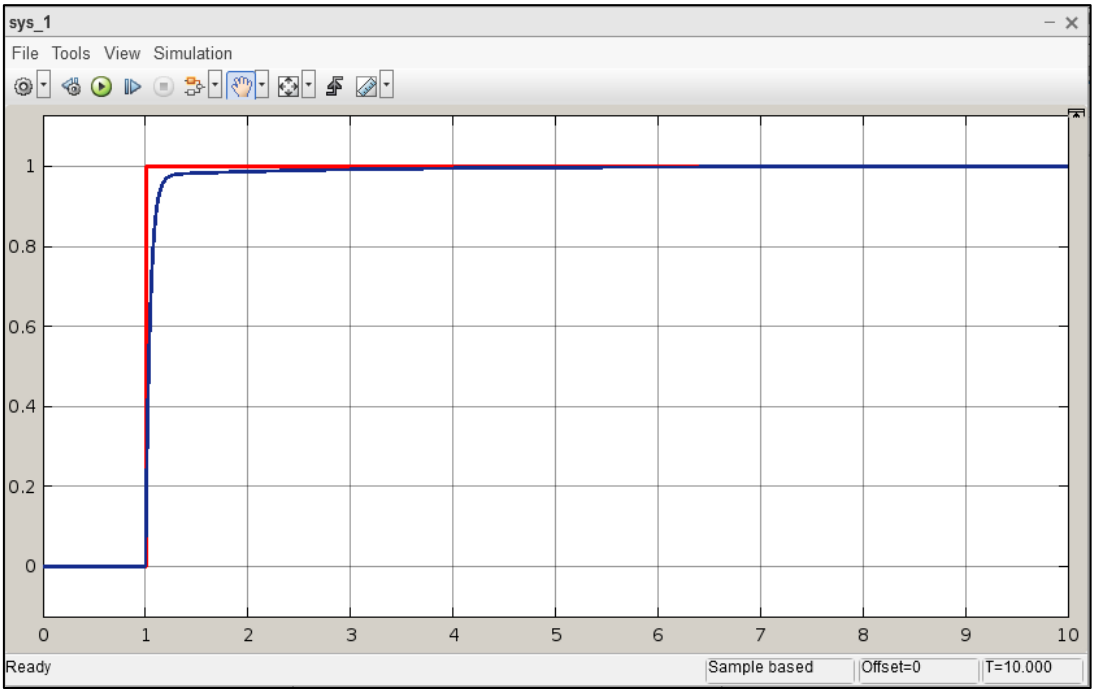

Figura 4 – Respuesta al escalón unitario Simulink – caso 1

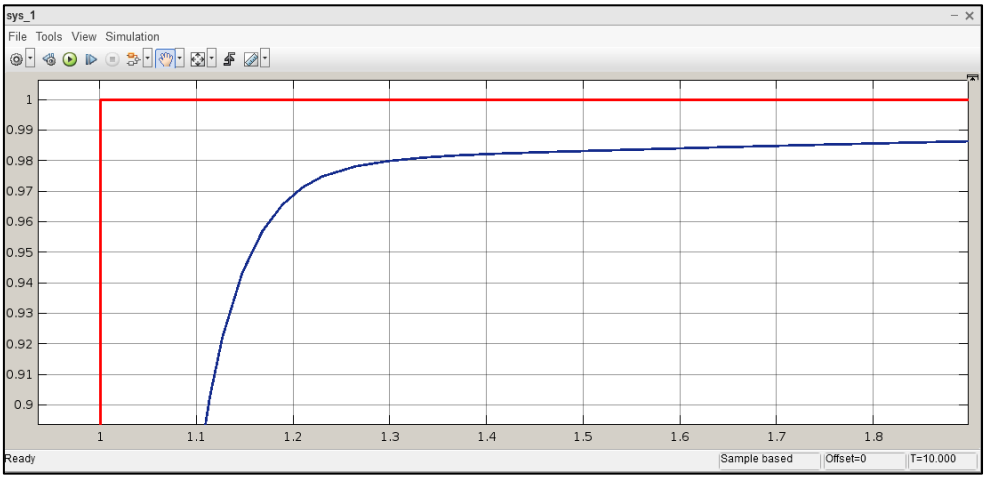

Figura 5 – Respuesta al escalón unitario Simulink (Zoom) – caso 1

Como se puede observar en la Figura 5, tras 0.3” la respuesta alcanza ese 98%. En este caso el 98% de un 1 es 0.98.

Script Matlab

```
%% Caso 1
I=3.1276;
numGs_1=[A ((A*B)/15)];
denGs_1=[1 C/7];
Gs_1= tf(numGs_1, denGs_1);

numRs_1=[I];
denRs_1=[1 0];
Rs_1=tf(numRs_1, denRs_1);

FT_BA_1=Rs_1*Gs_1;
FT_BC_1=feedback(FT_BA_1, 1);

step(FT_BC_1);
```

Caso 2- Continuo

Se pide un regulador para el siguiente sistema que cumpla: Error de velocidad nulo, sobreoscilación menor al 1.5% y tiempo de respuesta menor a 0.5".

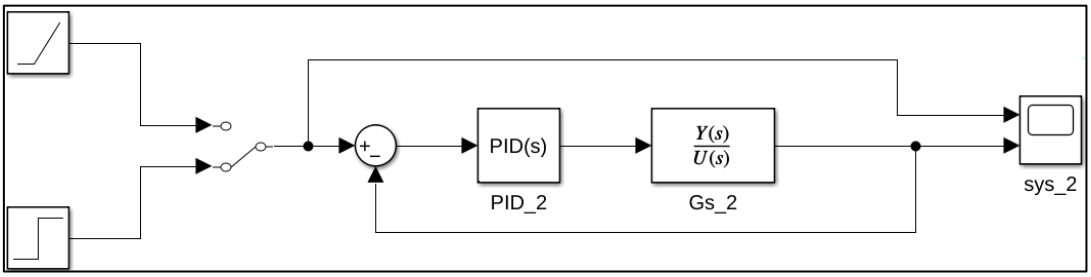

Figura 6 – Diagrama de bloques del sistema – caso 2

[Ec. 4] – Función de transferencia continua general – caso 2

$$G_{s_2} = \frac{F \cdot E}{s(s + \frac{E}{75})}$$

[Ec. 5] – Función de transferencia continua particular – caso 2

$$G_{s_2} = \frac{24}{s(s + 0.05333)}$$

En el Caso 2, la Fts de la planta **[Ec. 4]** es de 2º orden. Por lo tanto, sobreoscilación no será nula. Por otra parte, para conseguir un error de velocidad nulo, se ha de disponer de un doble integrador (polo doble en cero). Sin embargo, sólo se tiene un integrador en la planta, por lo tanto, se necesitaría otro.

Por ello, la primera estrategia de diseño sería probar como un regulador integrador (I), así se asegura el error de velocidad nulo. Lo único que faltaría por conseguir es ajustar la ganancia del regulador hasta alcanzar una sobreoscilación menor al 1.5% y un tiempo de asentamiento menor de 0.5".

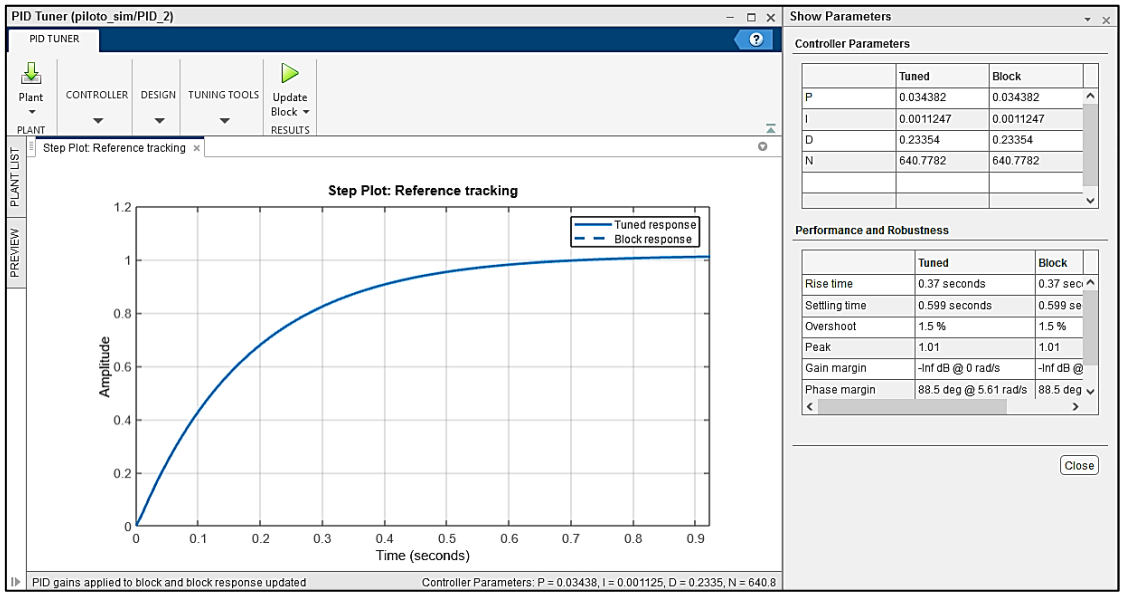

Figura 7 – Ajuste del regulador – caso 2

[Ec. 6] – Función de transferencia continua del Regulador – caso 2

$$R_{s_2} = \frac{149.7s^2 + 22.03s + 0.7202}{s(s + 640.8)}$$

Tras varios intentos con el tunner, no se ha podido conseguir las dos premisas de diseño. Por tanto, se ha tenido que declinarse por una premisa, quedándose la otra sujeta a la otra. Lo que se ha decidido es asegurar una sobreoscilación menor que el 1.5%, quedando un tiempo de asentamiento del 0.599, no tan lejano de lo exigido. Por otra parte, se ha observado que lo necesario para conseguir el ajuste del sistema era un regulador PID **[Ec. 6]**.

Evaluación del sistema

Excitamos el sistema con un escalón unitario, mediante la función **setp(…)** de Matlab.

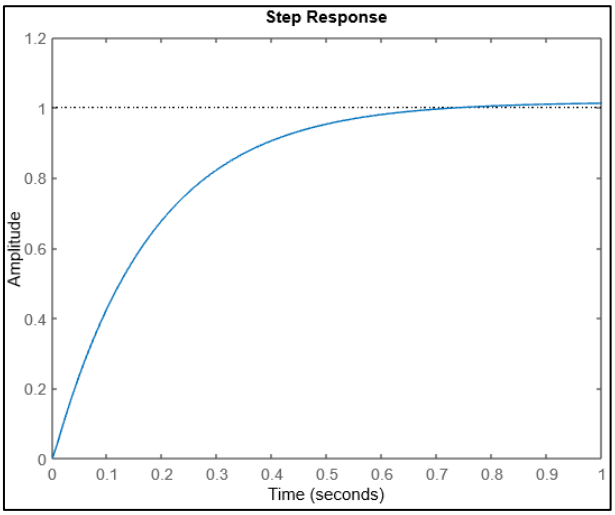

Figura 8 – Respuesta al escalón unitario – caso 2

En la Figura 8 se hace difícil observar y analizar el cumplimiento de las premisas de diseño, al menos en la versión online de Matlab. Por ello, se recurre a Simulink.

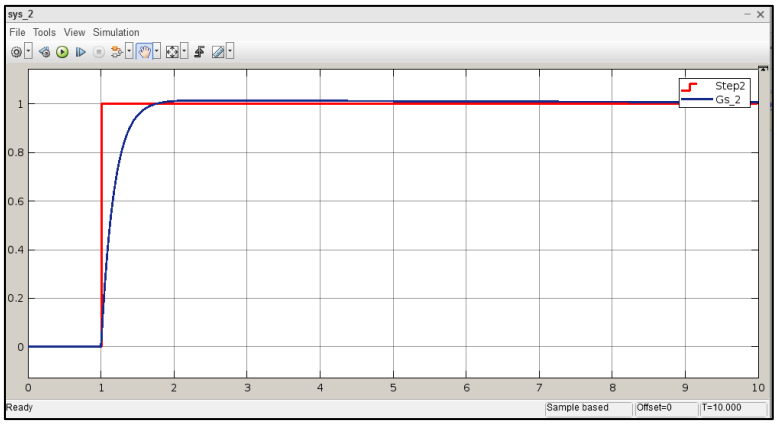

Figura 9 – Respuesta al escalón unitario Simulink – caso 2

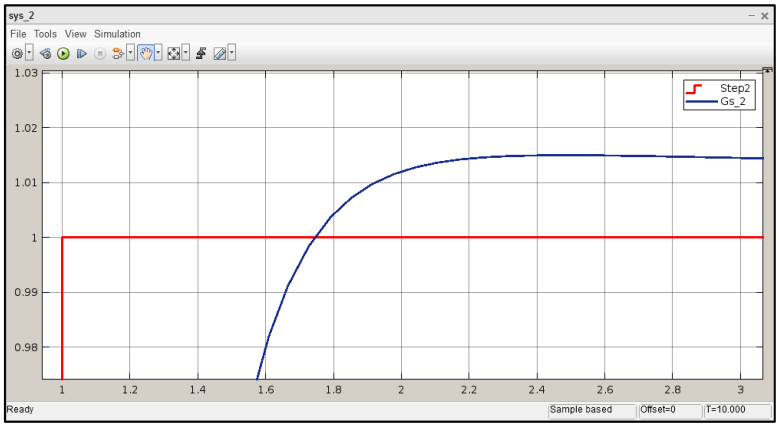

Figura 10 – Respuesta al escalón unitario Simulink (Zoom-1) – caso 2

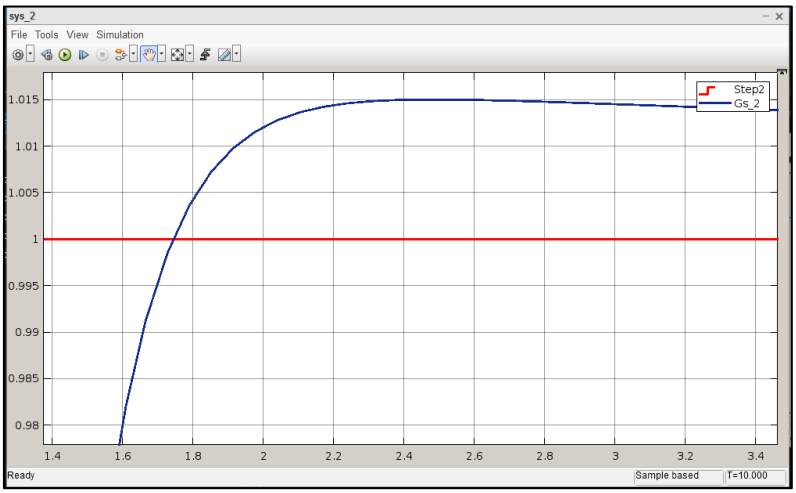

Figura 11 – Respuesta al escalón unitario Simulink (Zoom-2) – caso 2

Al excitar el sistema con una entrada escalón, se podría analizar el comportamiento dinámico del sistema. Por lo tanto, en la Figura 11, se observa que se consigue esa sobreoscilación menor al 1.5% (menor a 1.015), y en la Figura 10, se observa que el sistema entra en la banda de los ±0.02% en 0.6" (≈0.599).

Al excitar el sistema con una rampa unitaria, se hace difícil determinar con exactitud el error de velocidad por métodos gráficos (Figura 12). Por ello, no se sabe con certeza si realmente el error de velocidad es nulo cuando el tiempo tiende a infinito. Al hilo de lo anterior, se ha decidido comprobar mediante el teorema del valor final **[Ec. 9]** dicho error. Evaluando la función del error en el dominio de Laplace cuando la variable compleja **s** tiende a cero, se encuentra que el error de velocidad, es efectivamente nulo, como indica **[Ec. 9]**.

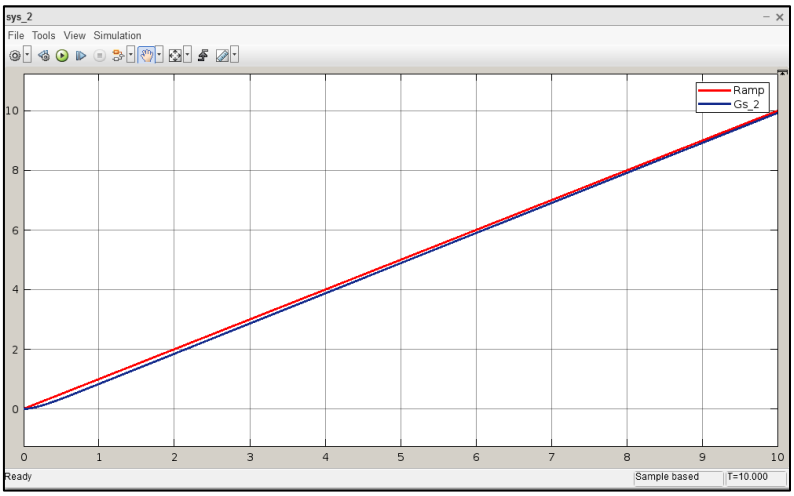

Figura 12 – Respuesta a la rampa unitaria Simulink – caso 2

Para comprobar el error de velocidad, se debe obtener del error. Si la excitación es la rampa unitaria ¡Error! No se encuentra el origen de la referencia.**]**: el error frente a dicha excitación queda reflejado en ¡Error! No se encuentra el origen de la referencia.**]**.

[Ec. 7] – Entrada tipo rampa unitaria en s – caso 2

$$X(s) = \frac{1}{s^2}$$

[Ec. 8] – Error en s – caso 2

$$E(s) = \frac{s^2 + 640.8s + 34.17}{s^4 + 640.8\,s^3 + 3626.0s^2 + 528.7s + 17.29}$$

[Ec. 9] – error de velocidad mediante el teorema del valor final - caso 2

$$\boldsymbol{ess_{v_2} = \lim_{s \to 0} s \cdot E(s) = 0.0}$$

Script Matlab

```
%% Caso 2
numGs_2=[F*E];
denGs_2=[1 E/75 0];
Gs_2= tf(numGs_2, denGs_2);

% PID
P_2=0.03438;
I_2=0.001124;
D_2=0.2335;
N_2=640.7782;

% Controller
numRs_2=[P_2+D_2*N_2 I_2+N_2*P_2 N_2*I_2];
denRs_2=[1 N_2 0];
Rs_2=tf(numRs_2, denRs_2);

FT_BA_2=Rs_2*Gs_2;
FT_BC_2=feedback(FT_BA_2, 1);

step(FT_BC_2);

Xs_2=tf([1],[1 0 0]);
Es_2=(1/(1+Gs_2*Rs_2))*Xs_2;

syms s
Es_2_s=getSymb_Gs(Es_2);
ess_2=vpa(limit(s*Es_2_s, s, 0), 4); % Error de velocidad
```

Caso 3 - Continuo

Se pide un regulador para el siguiente sistema que cumpla: Error de posición cero, sobreoscilación nula y tiempo de respuesta menor a 0.7”

[Ec. 10] – Función de transferencia continua general – caso 3

$$G_{s_3} = \frac{C \cdot E}{\left(s + \frac{E}{6}\right)\left(s + \frac{C}{5}\right)}$$

[Ec. 11] – Función de transferencia continua particular – caso 3

$$G_{s_3} = \frac{28}{s^2 + 2.067s + 0.9333}$$

En el Caso 3, la función de transferencia continua (Fts) de la planta **[Ec. 11]** es de 2º orden, y no tiene integradores. Por lo tanto, el sistema no tiene por qué tener sobreoscilación nula, ni cumplir con error de posición nulo.

Por ello, la primera se delega la estrategia de diseño al tunner, para conocer las componentes del PID que sobrevivirán, los cuales permitirán cumplir con los requisitos de diseño.

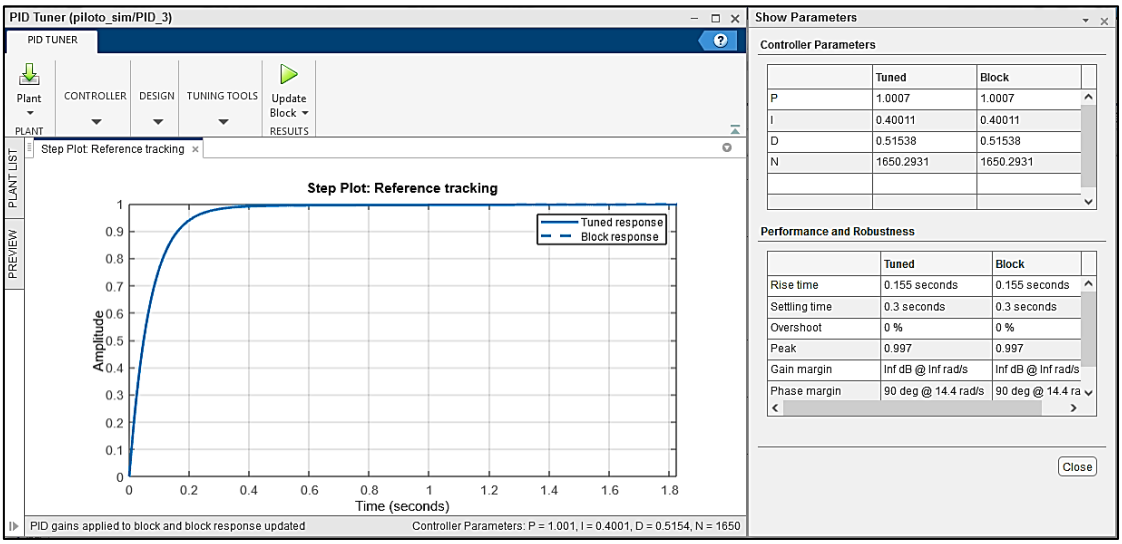

Figura 13 – Ajuste del regulador – caso 3

[Ec. 12] – Función de transferencia continua del Regulador – caso 3

$$R_{s_3} = \frac{851.5\, s^2 + 1652s + 660.3}{s(s + 1650)}$$

Tras conseguir el regulador **[Ec. 12]**, y así, el ajuste del sistema con el Tunner, en la tabla de **Show Parameters** (Figura 2), se puede ver que se ha conseguido una oscilación nula para un tiempo de asentamiento menor al 0.3”.

Evaluación del sistema

Excitamos el sistema con un escalón unitario, mediante la función **setp(…)** de Matlab.

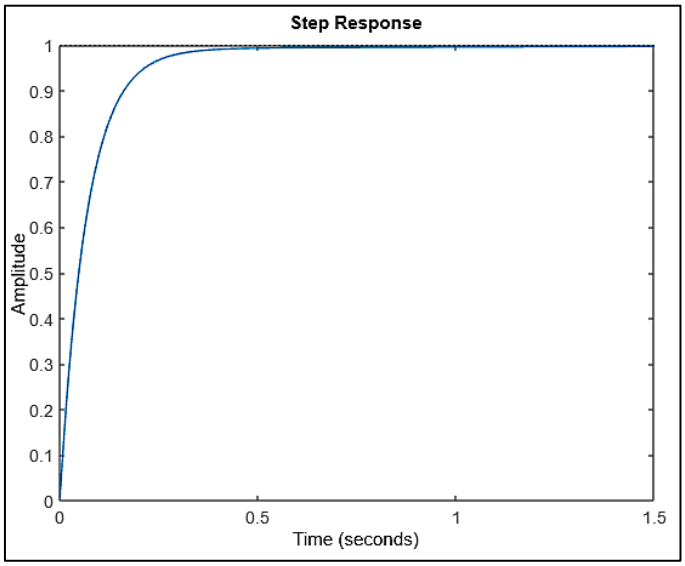

Figura 14 – Respuesta al escalón unitario – caso 3

En la Figura 3 se puede ver que al parecer la sobreoscilación es nula, así mismo el error de posición. No obstante, para comprobar con más exactitud, así como el tiempo de asentamiento, se procede a verlo con Simulink.

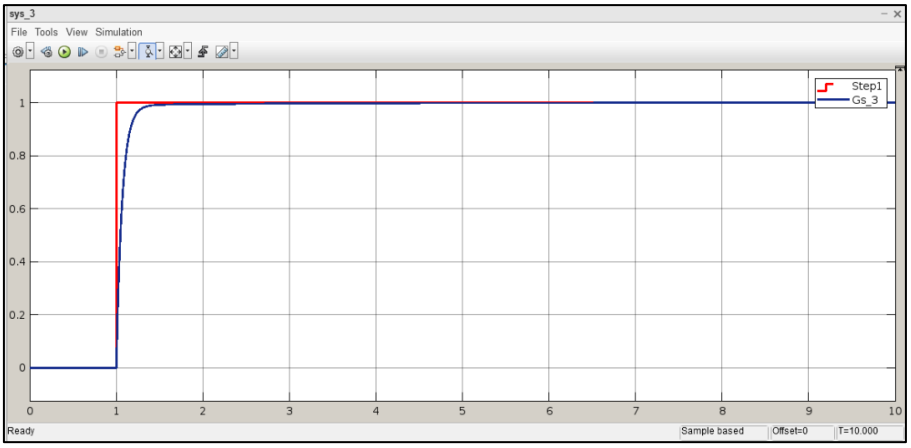

Figura 15 – Respuesta al escalón unitario Simulink – caso 3

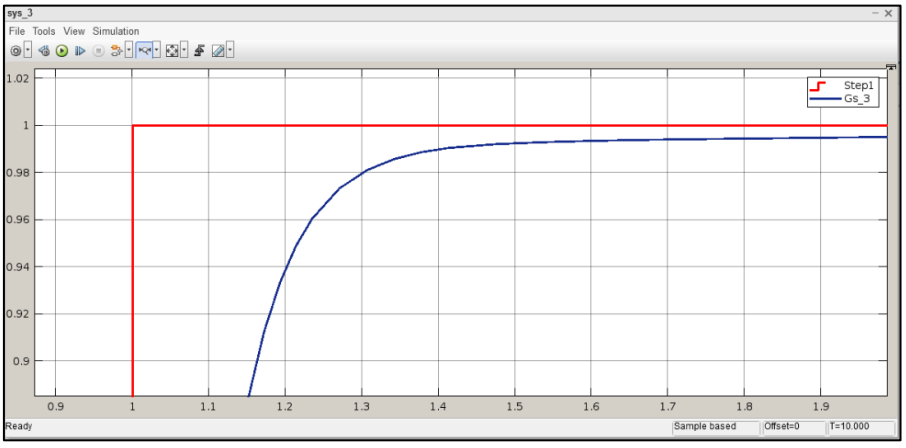

Figura 16 – Respuesta al escalón unitario Simulink (Zoom) – caso 3

Como se puede observar en la Figura 16, tras 0.3” la respuesta alcanza ese 98%. En este caso el 98%, siendo el sobreoscilación nula.

Script Matlab

```
%% Caso 3
numGs_3=[C*E];
denGs_3=[1 (C/5 + E/6) (C*E)/30];
Gs_3= tf(numGs_3, denGs_3);

% PID
P_3=1.00072465125397;
I_3=0.40011192003804;
D_3=0.515379643740182;
N_3=1650.29310619726;

% Controller
numRs_3=[P_3+D_3*N_3 I_3+N_3*P_3 N_3*I_3];
denRs_3=[1 N_3 0];
Rs_3=tf(numRs_3, denRs_3);

FT_BA_3=Rs_3*Gs_3;
FT_BC_3=feedback(FT_BA_3, 1);

step(FT_BC_3);

Xs_3=tf([1],[1 0]);
Es_3=(1/(1+Gs_3*Rs_3))*Xs_3;
```

APARTADO II

En este apartado se discretizarán los reguladores de Apartado I, para ello se vale de las instrucciones de discretización que proporciona Matlab. Después, se volverá a verificar los sistemas hallados como se hizo en el Apartado I.

Caso 1 – Discreto

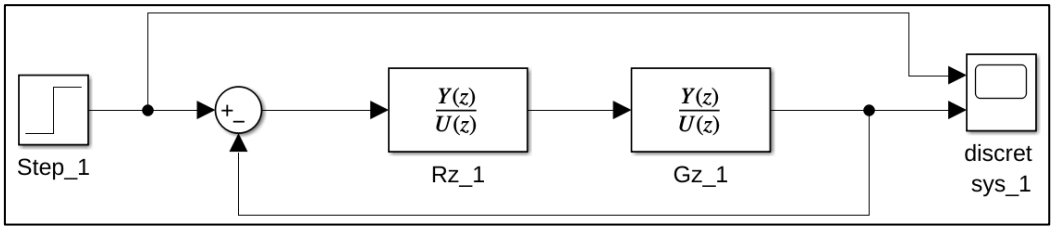

Figura 17 – Diagrama de bloques del sistema – caso 1 discreto

[Ec. 13] – Función de transferencia discretizada– caso 1 discreto

$$G_{z_1} = \frac{7z - 7}{z - 0.9999}$$

[Ec. 14] – Función de transferencia del Regulador discretizada– caso 1 discreto

$$R_{z_1} = \frac{0.0003128}{z - 1}$$

Evaluación del sistema

Excitamos el sistema discretizado con un escalón unitario, mediante la función **setp(…)** de Matlab.

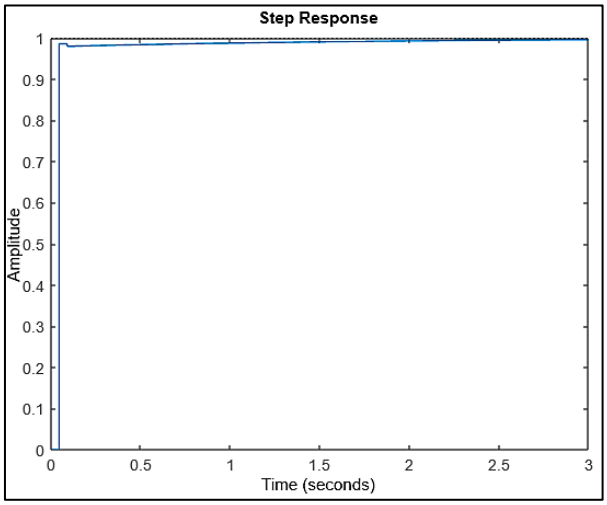

Figura 18 – Respuesta al escalón unitario – caso 1 discreto

En la Figura 18 se puede observar que se cumple que la sobreoscilación es nula. Ahora se procede con Simulink para comprobar que el tiempo de asentamiento sea menor que el 0,3".

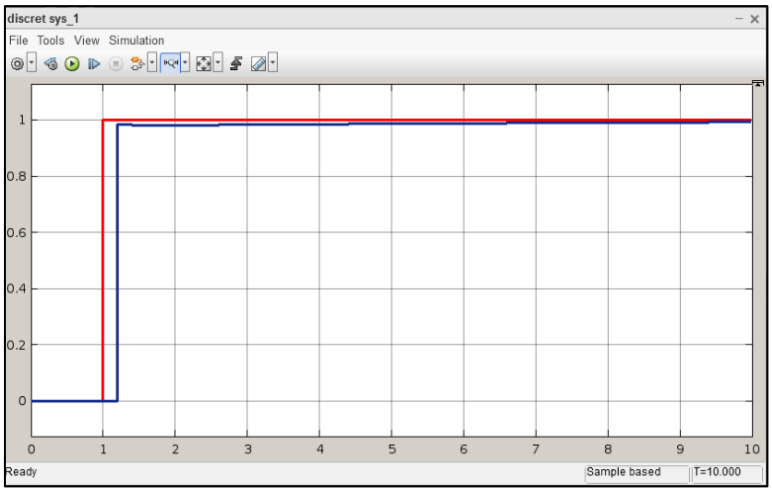

Figura 19 – Respuesta al escalón unitario Simulink – caso 1 discreto

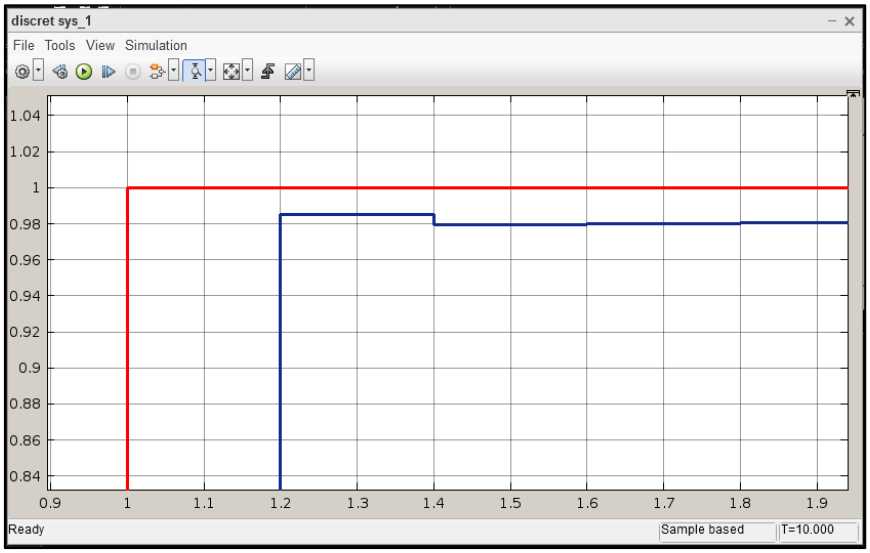

Figura 20 – Respuesta al escalón unitario Simulink (Zoom) – caso 1 discreto

Como se puede observar en la Figura 5, tras 0.2" la respuesta alcanza ese 98%. Siendo la sobreoscilación nula.

C/C++

En esta sección se realiza la implementación del regulador en C/C++. Por ello a partir de la expresión del regulador **[Ec. 14]**, se procede a hallar la ecuación en diferencias y la correspondiente implementación en C/C++.

[Ec. 15] – Función de transferencia del Regulador desarrollada– caso 1

$$R_{z_1} = \frac{U_z}{E_z} = \frac{0.0003128}{z-1}\left(\frac{z^{-1}}{z^{-1}}\right) = \frac{0.0003128z^{-1}}{1-z^{-1}} \rightarrow U_z = 0.0003128E_z z^{-1} + U_z z^{-1}$$

[Ec. 16] – Ecuación en diferencias del Regulador– caso 1

$$\boldsymbol{U_k = 0.0003128E_{k-1} + U_{k-1}}$$

Generación del código fuente C/C++

- ReguladorC1.h
- ReguladorC1.c

ReguladorC1.h

```
#ifndef REGULADOR_C1_H
#define REGULADOR_C1 _H

#define N   2
#define M   2

double controller(double E, double* b, double* a);
double reguladorC1(double Ek, double Ts, double time, double* aux);

#endif /*      REGULADOR_C1_H      */
```

ReguladorC1.c

```
#include "ReguladorC1.h"

double reguladorC1(double Ek, double Ts, double time, double* aux){
    static int k=0;
    static double Xk=0;
```

```
    double b[M]={0, 0.0003128};
    double a[N]={0, 1.0000000};

    if(time < Ts){
        k=0;
    }
    else if(time >= k*Ts){
        Xk=controller(Ek, b, a);
        k++;
        *aux=k*Ts;
    }
    return Xk;
}

double controller(double E, double* b, double* a){
    static int init_arrays = 0;
    static double U[N];
    static double E_values[M];

    if(!init_arrays){
        for(int i=0;i<N; i++) U[i]=0;
        for(int i=0;i<M; i++) E_values[i]=0;
        init_arrays = 1;
    }

    E_values[0]=E;
    double sum_Ek, sum_Uk;
    sum_Ek=0; sum_Uk=0;

    for(int k=0; k<M; k++)  sum_Ek = sum_Ek + b[k]*E_values[k];
    for(int k=0; k<N; k++)  sum_Uk = sum_Uk + a[k]*U[k];

    U[0]=sum_Ek+sum_Uk;
    for(int k=1; k<M; k++)  E_values[k]=E_values[k-1];
    for(int k=1; k<N; k++)  U[k]=U[k-1];

    return U[0];
}
```

Script Matlab

```
%% Discretizado
Gz_1=c2d(Gs_1, Ts);
numGz_1=cell2mat(Gz_1.Numerator);
denGz_1=cell2mat(Gz_1.Denominator);

Rz_1=c2d(Rs_1, Ts);
numRz_1=cell2mat(Rz_1.Numerator);
denRz_1=cell2mat(Rz_1.Denominator);

FTz_BA_1=Rz_1*Gz_1;
FTz_BC_1=feedback(FTz_BA_1, 1);
```

Caso 2 – Discreto

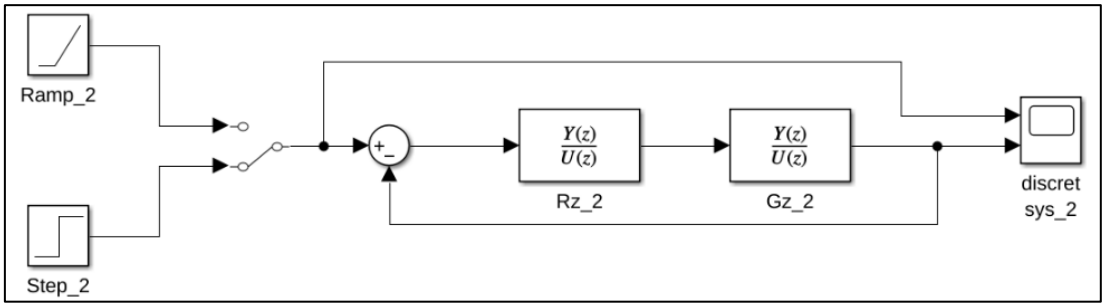

Figura 21 – Diagrama de bloques del sistema – caso 2 discreto

[Ec. 17] – Función de transferencia discretizada– caso 2 discreto

$$G_{z_2} = \frac{0.001465\,z\ +\ 0.001465}{z^2\ -\ 1.999\,z\ +\ 0.9994}$$

[Ec. 18] – Función de transferencia del regulador discretizado– caso 2 discreto

$$R_{z_2} = \frac{149.7\,z^2\ -\ 299.3\,z\ +\ 149.6}{z^2\ -\ 1.001\,z\ +\ 0.0008413}$$

Evaluación del sistema

Excitamos el sistema con un escalón unitario, mediante la función **setp(…)** de Matlab.

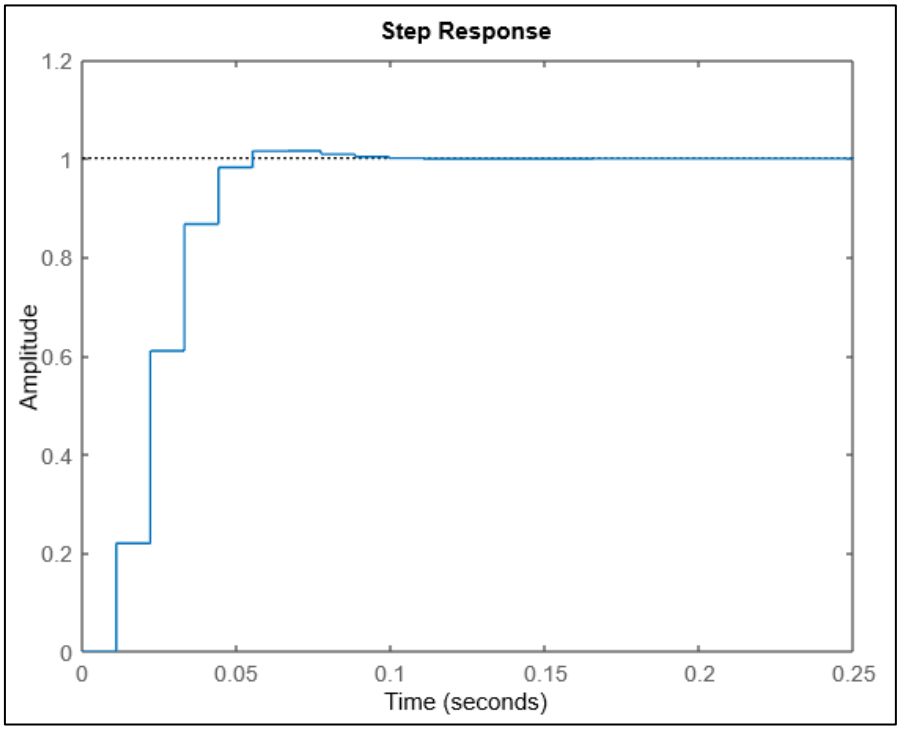

Figura 22 – Respuesta al escalón unitario – caso 2 discreto

En la Figura 22 se puede observar que al parecer si se cumplen las especificaciones de diseño, tanto en tiempo de asentamiento como en sobreoscilación. No obstante, se procede con los osciloscopios de Simulink, para verificar con más precisión el cumplimiento de dichas premisas.

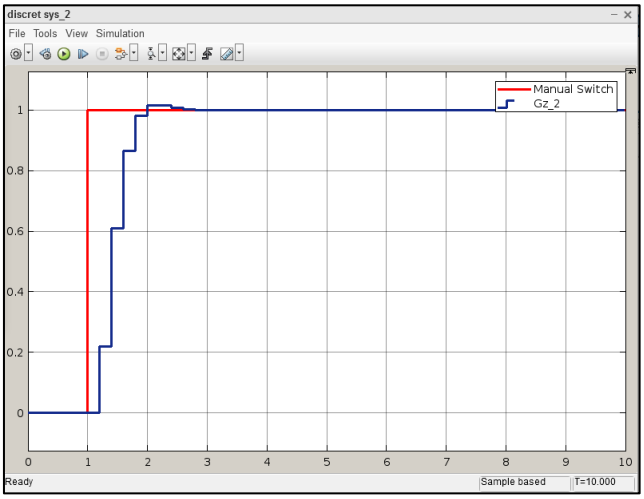

Figura 23 – Respuesta al escalón unitario Simulink – caso 2 discreto

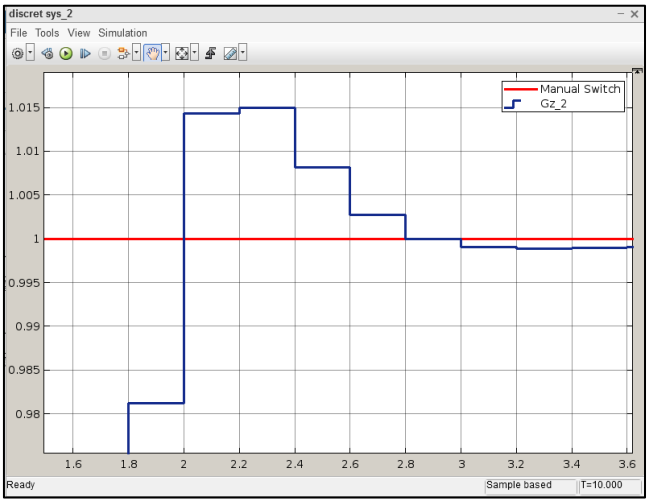

Figura 24 – Respuesta al escalón unitario Simulink (Zoom-1) – caso 2 discreto

Se observa que sí se cumple esa sobreoscilación de 1.5% sacrificando el tiempo de asentamiento que pasa de 0.5” (el requerido) a 0.8”, que es el mínimo tiempo de asentamiento que permite una sobreoscilación menor que el 1.5%.

Al excitar el sistema con una rampa unitaria, se observa que el error de velocidad no es nulo (Figura 25). Ello conlleva a pensar que la discretización ha sacrificado parte de las premisas de diseño, conseguidos en el tiempo continuo. No obstante, se procede a comprobar mediante el teorema del valor final discreto **[Ec. 21]** dicho error. Evaluando la función del error en el dominio Z cuando la variable compleja **z** tiende a 1. Efectivamente, el error de velocidad es nulo, así queda demostrado por ecuaciones que el error es nulo.

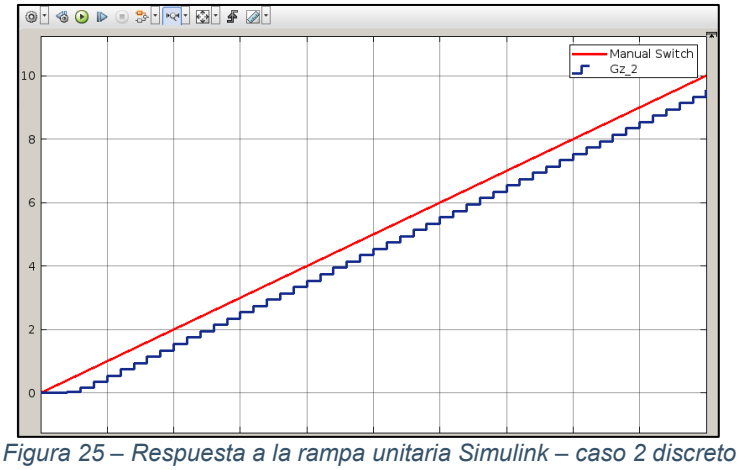

Figura 25 – Respuesta a la rampa unitaria Simulink – caso 2 discreto

[Ec. 19] – Entrada tipo rampa unitaria en Z – caso 2 discreto

$$X(z) = \frac{T_2 z^{-1}}{(1-z^{-1})^2} = \frac{0.01105z}{z^2 - 2z + 1}$$

[Ec. 20] – Error en Z – caso 2 discreto

$$E(z) = \frac{0.01105z^5 - 0.03315z^4 + 0.03316z^3 - 0.01107z^2 + 0.000009391z}{z^6 - 4.781z^5 + 9.344z^4 - 9.566z^3 + 5.444z^2 - 1.661z + 0.22}$$

[Ec. 21] – error de velocidad mediante el teorema del valor final - caso 2

$$\boldsymbol{ezz_{v_2} = \lim_{z\to 1}\left(\frac{z-1}{z}\right)\cdot E(z) = 0.0}$$

C/C++

En esta sección se realiza la implementación del regulador en C/C++. Por ello a partir de la expresión del regulador **[Ec. 18]**, se procede a hallar la ecuación en diferencias y la correspondiente implementación en C/C++.

[Ec. 22] – Función de transferencia del Regulador desarrollada– caso 2

$$R_{z_1} = \frac{U_z}{E_z} = \frac{149.7\,z^2 - 299.3\,z + 149.6}{z^2 - 1.001\,z + 0.0008413}\left(\frac{z^{-2}}{z^{-2}}\right) = \frac{149.7 - 299.3\,z^{-1} + 149.6z^{-2}}{1 - 1.001\,z^{-1} + 0.0008413z^{-2}}$$

$$\rightarrow \boldsymbol{U_z = 149.7E_z - 299.3E_z z^{-1} + 149.6E_z z^{-2} + 1.001U_z z^{-1} - 0.0008413U_z z^{-2}}$$

[Ec. 23] – Ecuación en diferencias del Regulador– caso 2

$$\boldsymbol{U_k = 149.7E_k - 299.3E_{k-1} + 149.6E_{k-2} + 1.001U_{k-1} - 0.0008413U_{k-2}}$$

Generación del código fuente C/C++

- ReguladorC2.h
- ReguladorC2.c

ReguladorC2.h

```
#ifndef REGULADOR_C2_H
#define REGULADOR_C2 _H

#define N   2
#define M   2

double controller(double E, double* b, double* a);
double reguladorC2(double Ek, double Ts, double time, double* aux);

#endif /*      REGULADOR_C2_H      */
```

ReguladorC2.c

```
#include "ReguladorC2.h"

double reguladorC2(double Ek, double Ts, double time, double* aux){
    static int k=0;
    static double Xk=0;

    double b[M]={149.7, -299.3, 149.6};
    double a[N]={0, 1.001, -0.0008413};

    if(time < Ts){
        k=0;
    }
    else if(time >= k*Ts){
        Xk=controller(Ek, b, a);
        k++;
        *aux=k*Ts;
    }
    return Xk;
```

```
}

double controller(double E, double* b, double* a){
    static int init_arrays = 0;
    static double U[N];
    static double E_values[M];

    if(!init_arrays){
        for(int i=0;i<N; i++) U[i]=0;
        for(int i=0;i<M; i++) E_values[i]=0;
        init_arrays = 1;
    }

    E_values[0]=E;
    double sum_Ek, sum_Uk;
    sum_Ek=0; sum_Uk=0;

    for(int k=0; k<M; k++)  sum_Ek = sum_Ek + b[k]*E_values[k];
    for(int k=0; k<N; k++)  sum_Uk = sum_Uk + a[k]*U[k];

    U[0]=sum_Ek+sum_Uk;
    for(int k=1; k<M; k++)  E_values[k]=E_values[k-1];
    for(int k=1; k<N; k++)  U[k]=U[k-1];

    return U[0];
}
```

Script Matlab

```
%% Discretizado
Ts_2=0.01105;

Gz_2=c2d(Gs_2, Ts_2);
numGz_2=cell2mat(Gz_2.Numerator);
denGz_2=cell2mat(Gz_2.Denominator);

Rz_2=c2d(Rs_2, Ts_2);
numRz_2=cell2mat(Rz_2.Numerator);
denRz_2=cell2mat(Rz_2.Denominator);

FTz_BA_2=Rz_2*Gz_2;
FTz_BC_2=feedback(FTz_BA_2, 1);

Xz_2=tf([Ts_2 0],[1 -2 1], Ts_2);
Ez_2=(1/(1+Gz_2*Rz_2))*Xz_2;

syms z
Ez_2_z=getSymb_Gz(Ez_2);
ezz_2=vpa(limit(((z-1)/z)*Ez_2_z, z, 1), 4);
```

Caso 3 - Discreto

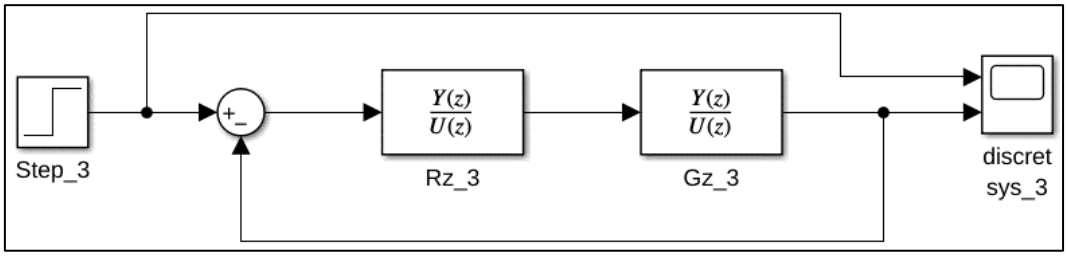

Figura 26 – Diagrama de bloques del sistema – caso 3 discreto

[Ec. 24] – Función de transferencia continua general – caso 3 discreto

$$G_{z_3} = \frac{0.001696\,z\ +\ 0.001684}{z^2 - 1.977z\ + 0.9774}$$

[Ec. 25] – Función de transferencia del regulador discretizado – caso 3 discreto

$$R_{z_3} = \frac{851.5\,z^2\ -\ 1702z\ +\ 850.5}{z^2 - 1.001z + \ 0.000842}$$

Evaluación del sistema

Excitamos el sistema con un escalón unitario, mediante la función **setp(…)** de Matlab.

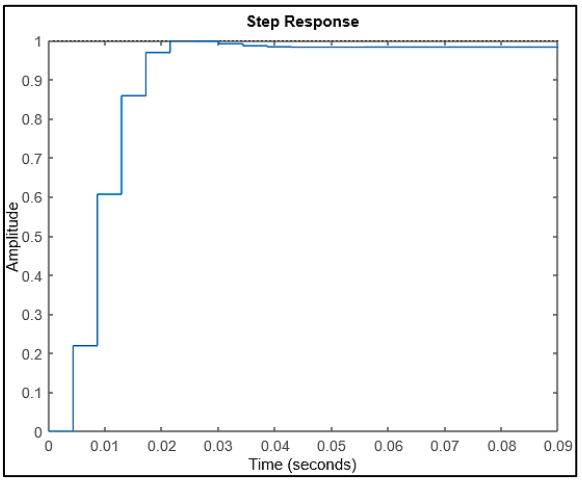

Figura 27 – Respuesta al escalón unitario – caso 3 discreto

En la Figura 27 se puede ver que al parecer la sobreoscilación es nula, así mismo el error de posición. No obstante, para comprobar todo ello con más exactitud, así como el tiempo de asentamiento, se procede a verlo con Simulink.

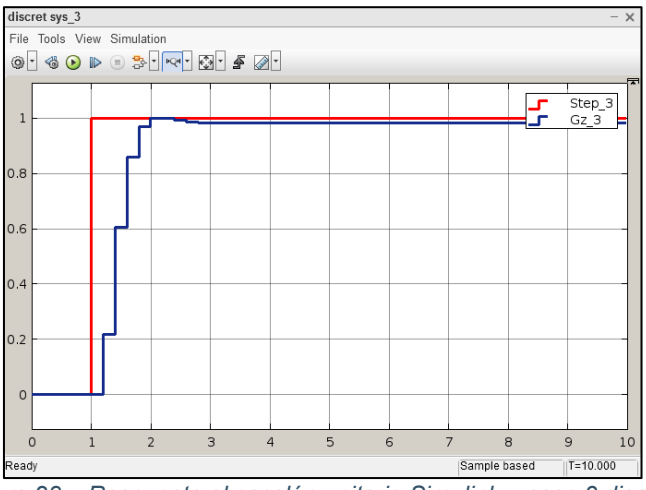

Figura 28 – Respuesta al escalón unitario Simulink – caso 3 discreto

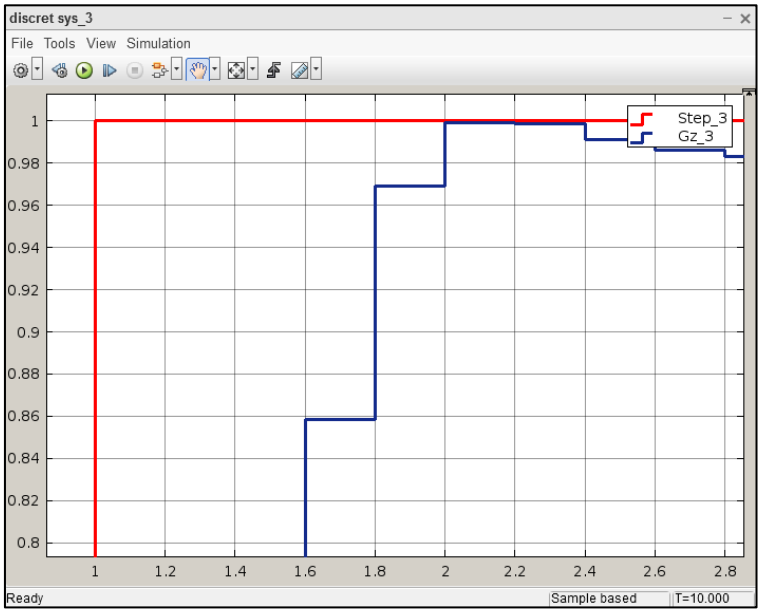

Figura 29 – Respuesta al escalón unitario Simulink (Zoom) – caso 3 discreto

Como se puede observar en la Figura 29, tras 1" la respuesta alcanza ese 98%. Siendo el sobreoscilación nula. Por lo que se puede observar es que se ha sacrificado ese tiempo de asentamiento en la discretización.

C/C++

En esta sección se realiza la implementación del regulador en C/C++. Por ello a partir de la expresión del regulador **[Ec. 25]**, se procede a hallar la ecuación en diferencias y la correspondiente implementación en C/C++.

[Ec. 26] – Función de transferencia del Regulador desarrollada– caso 2

$$R_{z_1} = \frac{U_z}{E_z} = \frac{851.5\,z^2 - 1702z + 850.5}{z^2 - 1.001z + 0.000842}\left(\frac{z^{-2}}{z^{-2}}\right) = \frac{851.5 - 1702z^{-1} + 850.5z^{-2}}{1 - 1.001z^{-1} + 0.000842z^{-2}}$$

$$\rightarrow \boldsymbol{U_z = 851.5E_z - 1702E_z\,z^{-1} + 850.5E_z z^{-2} + 1.001U_z z^{-1} - 0.000842U_z z^{-2}}$$

[Ec. 27] – Ecuación en diferencias del Regulador– caso 2

$$\boldsymbol{U_k = 851.5E_k - 1702E_{k-1} + 850.5E_{k-2} + 1.001U_{k-1} - 0.000842U_{k-2}}$$

Generación del código fuente C/C++

- ReguladorC3.h
- ReguladorC3.c

ReguladorC3.h

```
#ifndef REGULADOR_C3_H
#define REGULADOR_C3 _H

#define N   2
#define M   2

double controller(double E, double* b, double* a);
double reguladorC3(double Ek, double Ts, double time, double* aux);

#endif /*      REGULADOR_C3_H      */
```

ReguladorC3.c

```
#include "ReguladorC3.h"

double reguladorC3(double Ek, double Ts, double time, double* aux){
```

```
    static int k=0;
    static double Xk=0;

    double b[M]={851.5, -1702, 850.5};
    double a[N]={0, 1.001, -0.000842};

    if(time < Ts){
        k=0;
    }
    else if(time >= k*Ts){
        Xk=controller(Ek, b, a);
        k++;
        *aux=k*Ts;
    }
    return Xk;
}

double controller(double E, double* b, double* a){
    static int init_arrays = 0;
    static double U[N];
    static double E_values[M];

    if(!init_arrays){
        for(int i=0;i<N; i++) U[i]=0;
        for(int i=0;i<M; i++) E_values[i]=0;
        init_arrays = 1;
    }

    E_values[0]=E;
    double sum_Ek, sum_Uk;
    sum_Ek=0; sum_Uk=0;

    for(int k=0; k<M; k++)  sum_Ek = sum_Ek + b[k]*E_values[k];
    for(int k=0; k<N; k++)  sum_Uk = sum_Uk + a[k]*U[k];

    U[0]=sum_Ek+sum_Uk;
    for(int k=1; k<M; k++)  E_values[k]=E_values[k-1];
    for(int k=1; k<N; k++)  U[k]=U[k-1];

    return U[0];
}
```

Script Matlab

```
%% Discretizado
Ts_3=0.00429;

Gz_3=c2d(Gs_3, Ts_3);
numGz_3=cell2mat(Gz_3.Numerator);
denGz_3=cell2mat(Gz_3.Denominator);

Rz_3=c2d(Rs_3, Ts_3);
numRz_3=cell2mat(Rz_3.Numerator);
denRz_3=cell2mat(Rz_3.Denominator);

FTz_BA_3=Rz_3*Gz_3;
FTz_BC_3=feedback(FTz_BA_3, 1);
```

APARTADO III

En este apartado, empleando el tunner, se diseñarán reguladores discretos (P, PI, PD o PID) que satisfagan las especificaciones solicitadas en cada caso.

Caso 1 – Discreto

Se pretende que el sistema tenga error de velocidad nulo y un tiempo de respuesta inferior a 2 segundos y sobreoscilación inferior al 15%.

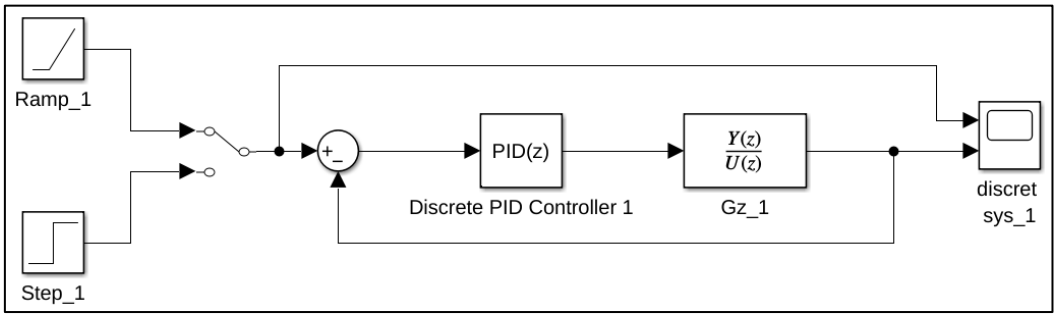

Figura 30 – Diagrama de bloques del sistema – caso 1

[Ec. 28] – Función de transferencia continua general – caso 1

$$G_{s_1} = \frac{\frac{C}{B}}{s\left(s + \frac{C}{9}\right)}$$

[Ec. 29] – Función de transferencia continua particular – caso 1

$$G_{s_1} = \frac{0.875}{s(s + 0.7778)}$$

[Ec. 30] – Función de transferencia discretizada– caso 1

$$G_{z_1} = \frac{0.015\,z\, +\, 0.01428}{z^2 - 1.863z + 0.8628}$$

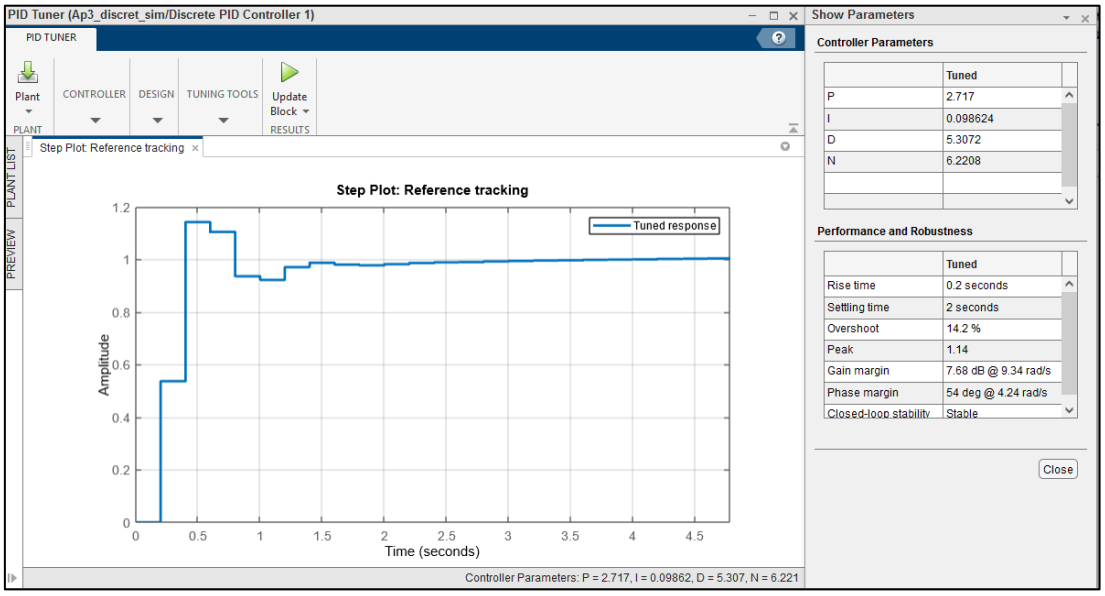

Figura 31 – Ajuste del Regulador discreto – caso 1

[Ec. 31] – Función de transferencia discreta del Regulador – caso 1

$$R_{z_1} = \frac{35.73216z^2 - 67.7425z + 32.61866}{z^2 - 0.8199z - 0.18}$$

En la **[Ec. 31]** se muestra el regulador obtenido con el tunner. Éste tiene un polo en z=1, lo que garantiza error de posición nulo. Además, la planta **[Ec. 29]**, tiene un polo en z=1. Por lo tanto la función de transferencia en bucle abierto del sistema (FT_{BA}) tiene doble integrador (doble polo en z=1), y por tanto se cumple con la premisa de error de velocidad nulo.

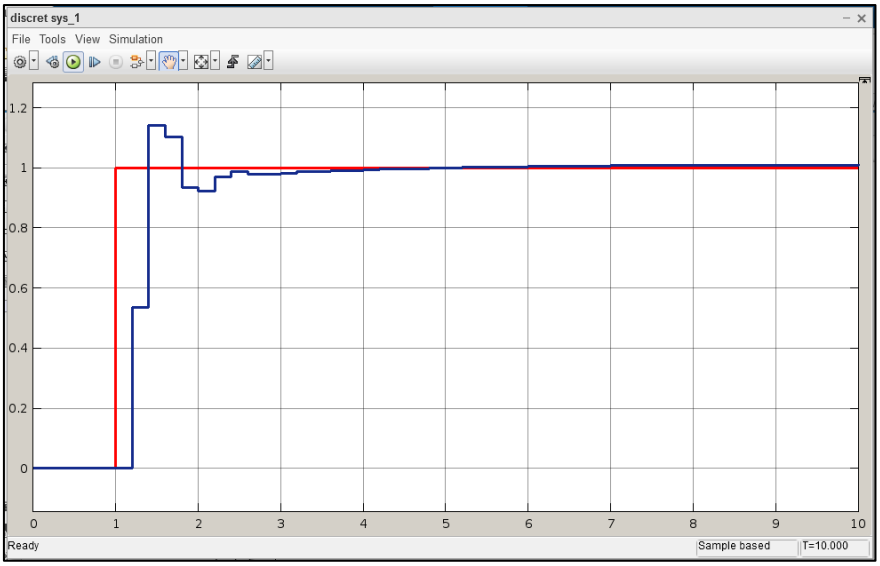

Figura 32 – Respuesta al escalón unitario Simulink – caso 1

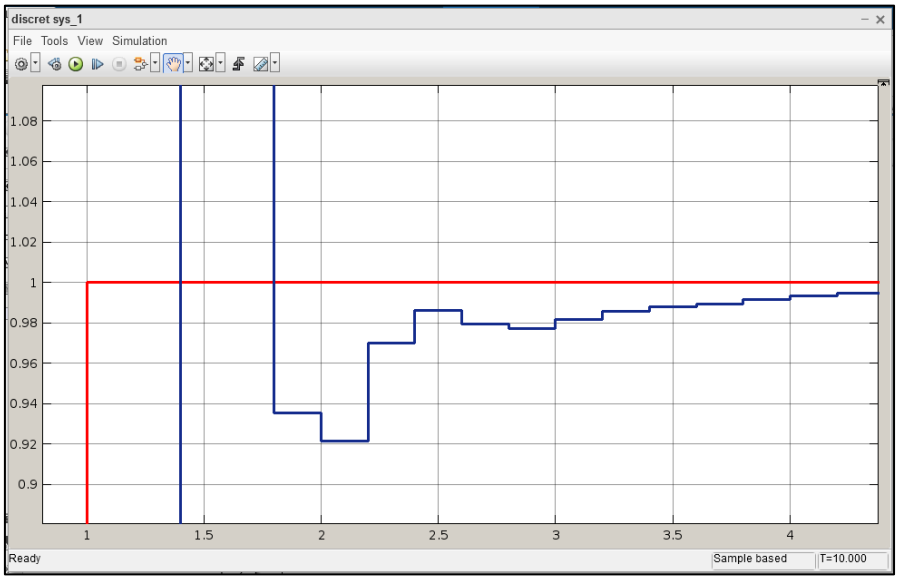

Figura 33 – Respuesta al escalón unitario Simulink (Zoom-1) – caso 1

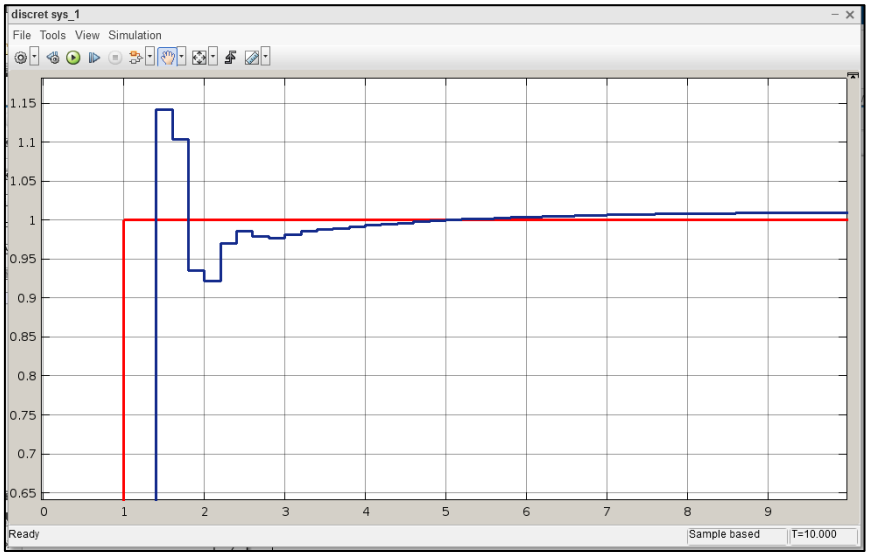

Figura 34 – Respuesta al escalón unitario Simulink (Zoom-2) – caso 1

En las figuras anteriores, se puede analizar el comportamiento dinámico del sistema. Así en la Figura 33 se observa que se cumple con el tiempo de asentamiento de 2”. Así mismo en la Figura 34 se cumple con esa sobreoscilación menor que el 15%.

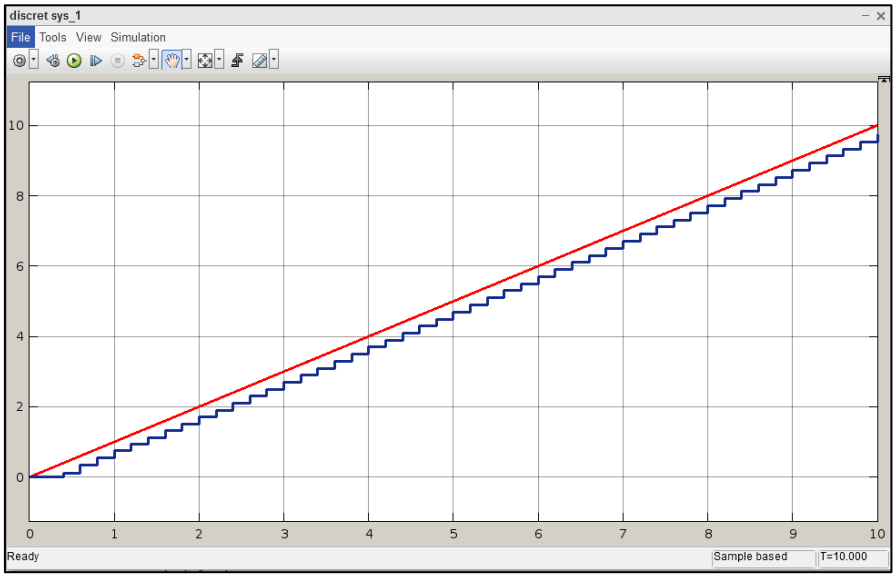

Figura 35 – Respuesta a la rampa unitaria Simulink – caso 1

En la Figura 35 no factible observar un error de velocidad nulo. No obstante, el doble integrador garantiza un error de velocidad nulo en el infinito.

Caso 2 – Discreto

Se espera que el sistema tenga error de posición cero, tiempo de respuesta menor a 0.5 segundos y una sobreoscilación nula.

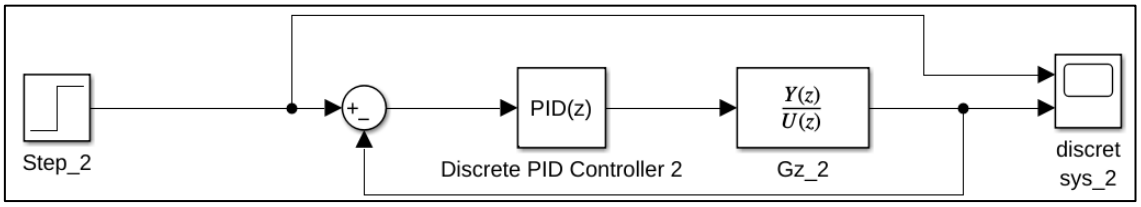

Figura 36 – Diagrama de bloques del sistema – caso 2

[Ec. 32] – Función de transferencia continua general – caso 2

$$G_{s_2} = \frac{3C}{s^2 + 5Es + 0.1C}$$

[Ec. 33] – Función de transferencia continua particular – caso 2

$$G_{s_2} = \frac{21}{s^2 + 20s + 0.7}$$

[Ec. 34] – Función de transferencia discretizada– caso 2

$$G_{z_2} = \frac{0.01931\,z\, +\, 0.01387}{z^2 -\, 1.367\,z\, +\, 0.3679}$$

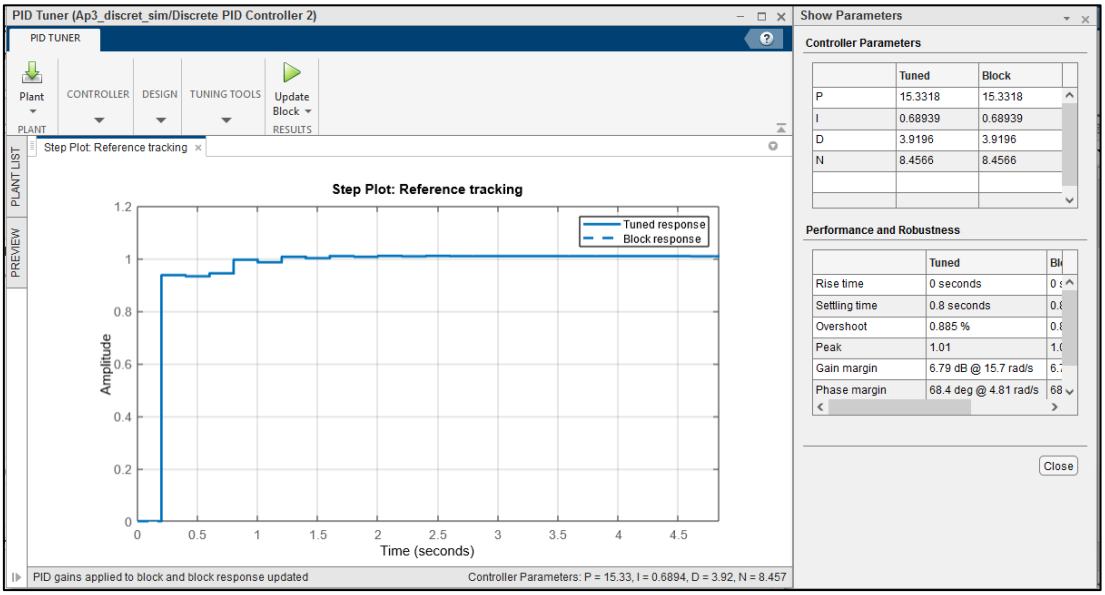

Figura 37 – Ajuste del Regulador discreto – caso 1

[Ec. 35] – Función de transferencia discreta del Regulador – caso 1

$$R_{z_2} = \frac{30.695z^2 - 58.3508z + 27.6626}{z^2 - 1.804z + 0.804}$$

En la **[Ec. 35]** se muestra el regulador obtenido con el tunner. Éste tiene un polo en z=1, lo que garantiza error de posición nulo. No obstante, el único que se ha podido conseguir con el tunner es un regulador con una sobreoscilación del 0.885% y un tiempo de asentamiento del 0.8” (Figura 37), dado que el sistema es de segundo orden, es realmente difícil cumplir con una sobreoscilación nula con un tiempo de asentamiento tan pequeño como es 0.5”. Por lo tanto, es factible concluir que se ha podido llegar a un compromiso adecuado, en la medida de lo posible.

Caso 3 – Discreto

Se espera que el sistema tenga error de velocidad cero, tiempo de respuesta inferior a 2 segundos y sobreoscilación nula.

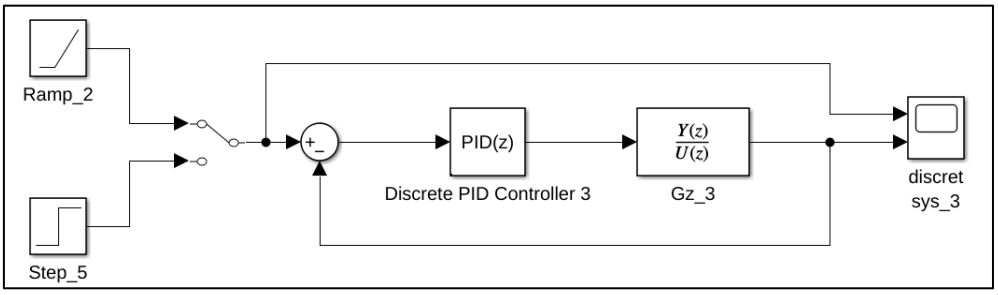

Figura 38 – Diagrama de bloques del sistema – caso 3

[Ec. 36] – Función de transferencia continua general – caso 3

$$G_{s_3} = \frac{5D}{s + 15B}$$

[Ec. 37] – Función de transferencia continua particular – caso 3

$$G_{s_3} = \frac{10}{s + 120}$$

[Ec. 38] – Función de transferencia discretizada– caso 3

$$G_{z_3} = \frac{0.08333}{z - 3.775 \cdot 10^{-11}}$$

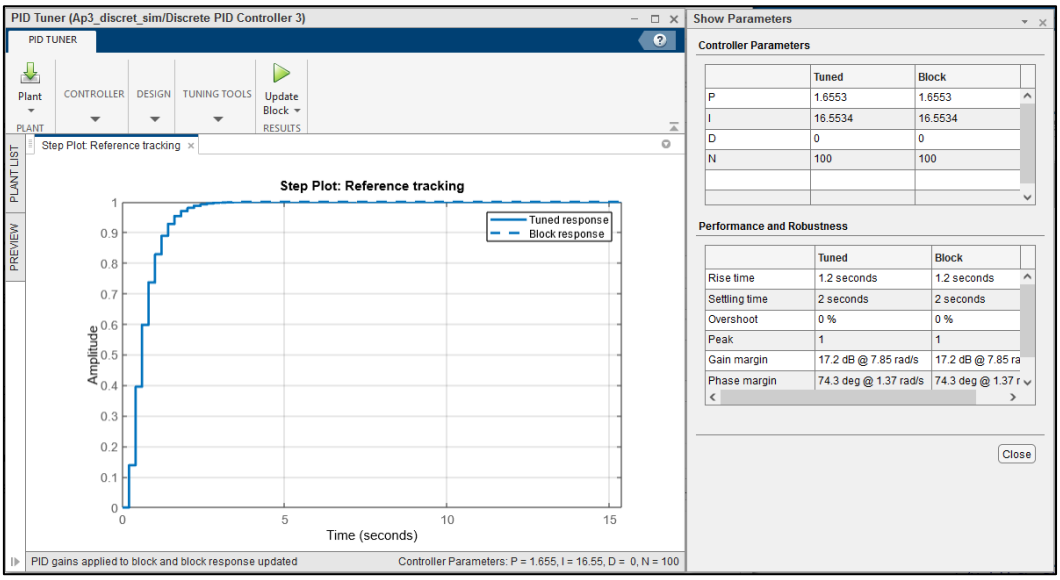

Figura 39 – Ajuste del Regulador discreto – caso 3

[Ec. 39] – Función de transferencia discreta del Regulador – caso 3

$$R_{z_3} = \frac{1.6553z + 1.65538}{z - 1}$$

En la **[Ec. 39]** se muestra el regulador obtenido con el tunner. Éste tiene un polo en z=1, lo que garantiza error de posición nulo.

Caso 4 – Discreto

Se espera que el sistema tenga error de velocidad cero, tiempo de respuesta inferior a 2 segundos y sobreoscilación nula.

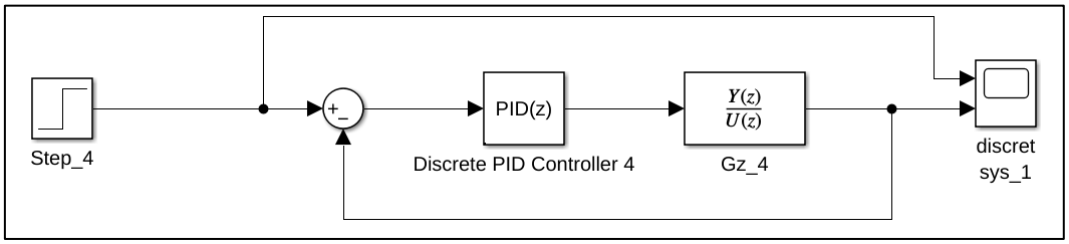

Figura 40 – Diagrama de bloques del sistema – caso 4

[Ec. 40] – Función de transferencia continua general – caso 4

$$G_{s_4} = \frac{6D(s+5B)}{(s+25F)(s+7E)}$$

[Ec. 41] – Función de transferencia continua particular – caso 4

$$G_{s_4} = \frac{12s+480}{s^2+178s+4200}$$

[Ec. 42] – Función de transferencia discretizada– caso 4

$$G_{z_4} = \frac{0.1142z - 0.0002135}{z^2 - 0.00296z + 8.414 \cdot 10^{-17}}$$

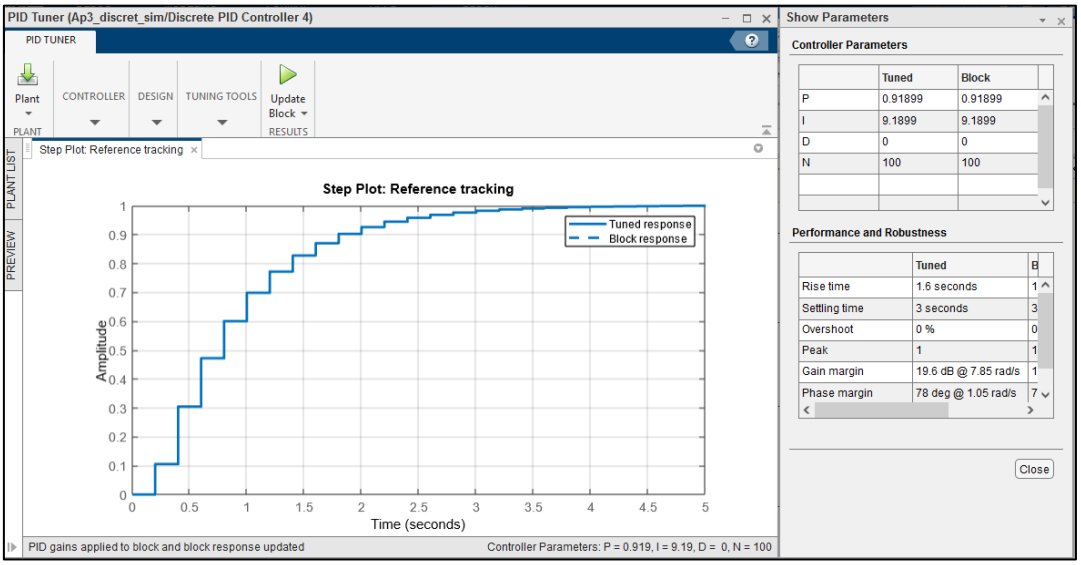

Figura 41 – Ajuste del Regulador discreto – caso 4

[Ec. 43] – Función de transferencia discreta del Regulador – caso 4

$$R_{z_4} = \frac{0.9189z - 0.9189}{z-1}$$

En la **[Ec. 43]** se muestra el regulador obtenido con el tunner. Éste tiene un polo en z=1, lo que garantiza error de posición nulo. Así mismo, se pude observar en la tabla de parámetros (Figura 41) que se consiguen las premisas de diseño.

ANEXOS

Script Matlab- Apartado I y Apartado II

```
%% En nombre de ALLAH,

%% NIA=787246
A=7; B=8; C=7; D=2; E=4; F=6;

%% Caso 1
I=3.1276;
numGs_1=[A ((A*B)/15)];
denGs_1=[1 C/7];
Gs_1= tf(numGs_1, denGs_1);

numRs_1=[I];
denRs_1=[1 0];
Rs_1=tf(numRs_1, denRs_1);

Xs_1=tf([1],[1 0]);
Es_1=(1/(1+Gs_1*Rs_1))*Xs_1;

FT_BA_1=Rs_1*Gs_1;
FT_BC_1=feedback(FT_BA_1, 1);

step(FT_BC_1);

syms s
Es_1_s=getSymb_Gs(Es_1);
ess_1=vpa(limit(s*Es_1_s, s, 0), 4);

%% Discretizado
Ts_1=0.045;

Gz_1=c2d(Gs_1, Ts_1);
numGz_1=cell2mat(Gz_1.Numerator);
denGz_1=cell2mat(Gz_1.Denominator);

Rz_1=c2d(Rs_1, Ts_1);
numRz_1=cell2mat(Rz_1.Numerator);
denRz_1=cell2mat(Rz_1.Denominator);

FTz_BA_1=Rz_1*Gz_1;
FTz_BC_1=feedback(FTz_BA_1, 1);

%% Caso 2
numGs_2=[F*E];
denGs_2=[1 E/75 0];
Gs_2= tf(numGs_2, denGs_2);

% PID
P_2=0.03438;
I_2=0.001124;
D_2=0.2335;
N_2=640.7782;

% Controller
numRs_2=[P_2+D_2*N_2 I_2+N_2*P_2 N_2*I_2];
denRs_2=[1 N_2 0];
Rs_2=tf(numRs_2, denRs_2);

FT_BA_2=Rs_2*Gs_2;
FT_BC_2=feedback(FT_BA_2, 1);
```

```
step(FT_BC_2);

Xs_2=tf([1],[1 0 0]);
Es_2=(1/(1+Gs_2*Rs_2))*Xs_2;

syms s
Es_2_s=getSymb_Gs(Es_2);
ess_2=vpa(limit(s*Es_2_s, s, 0), 4);

%% Discretizado
Ts_2=0.01105;

Gz_2=c2d(Gs_2, Ts_2);
numGz_2=cell2mat(Gz_2.Numerator);
denGz_2=cell2mat(Gz_2.Denominator);

Rz_2=c2d(Rs_2, Ts_2);
numRz_2=cell2mat(Rz_2.Numerator);
denRz_2=cell2mat(Rz_2.Denominator);

FTz_BA_2=Rz_2*Gz_2;
FTz_BC_2=feedback(FTz_BA_2, 1);

Xz_2=tf([Ts_2 0],[1 -2 1], Ts_2);
Ez_2=(1/(1+Gz_2*Rz_2))*Xz_2;

syms z
Ez_2_z=getSymb_Gz(Ez_2);
ezz_2=vpa(limit(((z-1)/z)*Ez_2_z, z, 1), 4);

%% Caso 3
numGs_3=[C*E];
denGs_3=[1 (C/5 + E/6) (C*E)/30];
Gs_3= tf(numGs_3, denGs_3);

% PID
P_3=1.00072465125397;
I_3=0.40011192003804;
D_3=0.515379643740182;
N_3=1650.29310619726;

% Controller
numRs_3=[P_3+D_3*N_3 I_3+N_3*P_3 N_3*I_3];
denRs_3=[1 N_3 0];
Rs_3=tf(numRs_3, denRs_3);

FT_BA_3=Rs_3*Gs_3;
FT_BC_3=feedback(FT_BA_3, 1);

step(FT_BC_3);

Xs_3=tf([1],[1 0]);
Es_3=(1/(1+Gs_3*Rs_3))*Xs_3;

%% Discretizado
Ts_3=0.00429;

Gz_3=c2d(Gs_3, Ts_3);
numGz_3=cell2mat(Gz_3.Numerator);
denGz_3=cell2mat(Gz_3.Denominator);

Rz_3=c2d(Rs_3, Ts_3);
```

```
numRz_3=cell2mat(Rz_3.Numerator);
denRz_3=cell2mat(Rz_3.Denominator);

FTz_BA_3=Rz_3*Gz_3;
FTz_BC_3=feedback(FTz_BA_3, 1);

%% © NAOUFAL EL RHAZZALI
```

Script Matlab- Apartado III

```
%% En nombre de ALLAH,

%% NIA=787246
A=7; B=8; C=7; D=2; E=4; F=6;

%% Caso 1
numGs_1=[C/B];
denGs_1=[1 C/9 0];
Gs_1= tf(numGs_1, denGs_1);

%% Discretizado
SO_1=15; % 15%
tr_1=2; % 2s
gi_1=gifor_SO(SO_1);
Ts_1=get_samplingTime(tr_1, gi_1);

Gz_1=c2d(Gs_1, Ts_1);
numGz_1=cell2mat(Gz_1.Numerator);
denGz_1=cell2mat(Gz_1.Denominator);

%% Caso 2
numGs_2=[3*C];
denGs_2=[1 5*E 0.1*C];
Gs_2= tf(numGs_2, denGs_2);

%% Discretizado
SO_2=0.00001; % 0%
tr_2=0.5; % 0.5s
gi_2=gifor_SO(SO_2);
Ts_2=get_samplingTime(tr_2, gi_2);

Gz_2=c2d(Gs_2, Ts_2);
numGz_2=cell2mat(Gz_2.Numerator);
denGz_2=cell2mat(Gz_2.Denominator);

%% Caso 3
numGs_3=[5*D];
denGs_3=[1 15*B];
Gs_3= tf(numGs_3, denGs_3);

%% Discretizado
SO_3=0.00001; % 0%
tr_3=2; % 2s
gi_3=gifor_SO(SO_3);
Ts_3=get_samplingTime(tr_3, gi_3);

Gz_3=c2d(Gs_3, Ts_3);
numGz_3=cell2mat(Gz_3.Numerator);
denGz_3=cell2mat(Gz_3.Denominator);

%% Caso 4
numGs_4=[6*D 30*D*B];
denGs_4=[1 25*F+7*E 175*E*F];
Gs_4= tf(numGs_4, denGs_4);

%% Discretizado
SO_4=25; % 25%
tr_4=3; % 3s
gi_4=gifor_SO(SO_4);
Ts_4=get_samplingTime(tr_4, gi_4);
```

```
Gz_4=c2d(Gs_4, Ts_4);
numGz_4=cell2mat(Gz_4.Numerator);
denGz_4=cell2mat(Gz_4.Denominator);

%% © NAOUFAL EL RHAZZALI
```

Matlab- Funciones propias desarrolladas

get_samplingTime()

```
function [T] = get_samplingTime(tr, dampingFactor)
%% Sampling time calculationing
    % First method
    % Sampling time:
    T1=tr/10;

    % Second method
    % ζ = dampingFactor
    % tr = establishment time

    % sigma = σ
    sigma = 4/tr;
    % wn = parte real del polo (Hz), y fctAmortg = ζ
    wn = sigma/dampingFactor;
    % wd = parte imaginaria del polo (Hz)
    wd=wn*sqrt(1-((dampingFactor)^2));
    % Sampling time:
    T2=pi/(5*wd);

    % The optimal sampling time
    T=min(T1, T2);
end
```

getSymb_Gs()

```
function Gs_s = getSymb_Gs(Gs_tf)
    syms s

    num=cell2mat(Gs_tf.Numerator);
    den=cell2mat(Gs_tf.Denominator);

    num_len=length(num);
    den_len=length(den);

    num_s=0;
    den_s=0;

    for i=1:1:num_len
        num_s=num_s+num(i)*s^(num_len-i);
    end

    for i=1:1:den_len
        den_s=den_s+den(i)*s^(den_len-i);
    end

    Gs_s=num_s/den_s;
    Gs_s=vpa(Gs_s, 4);
end
```

getSymb_Gz()

```
function Gz_z = getSymb_Gz(Gz_tf)
    syms z

    num=cell2mat(Gz_tf.Numerator);
    den=cell2mat(Gz_tf.Denominator);

    num_len=length(num);
    den_len=length(den);
```

```
    num_z=0;
    den_z=0;

    for i=1:1:num_len
        num_z=num_z+num(i)*z^(num_len-i);
    end

    for i=1:1:den_len
        den_z=den_z+den(i)*z^(den_len-i);
    end

    Gz_z=num_z/den_z;
    Gz_z=vpa(Gz_z, 4);
end
```

gifor_SO()

```
function [gi] = gifor_SO(SO)
    gi = sqrt(((log(SO/100.0)^2))/((pi^2)+(log(SO/100.0)^2)));
end
```