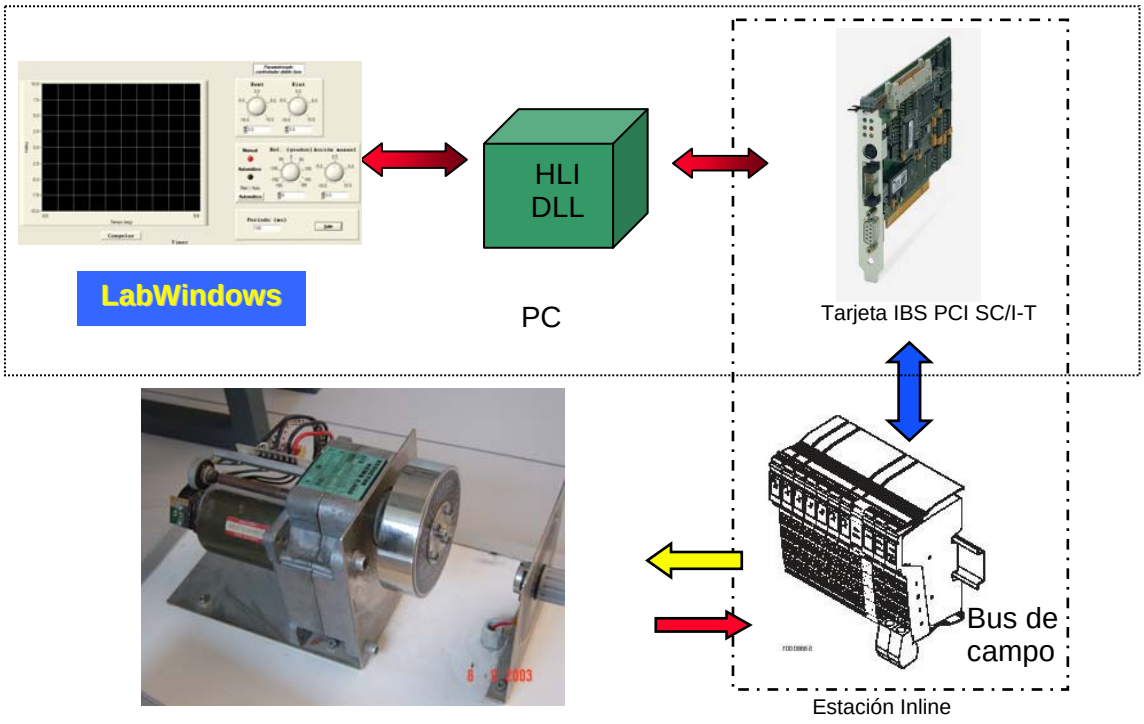
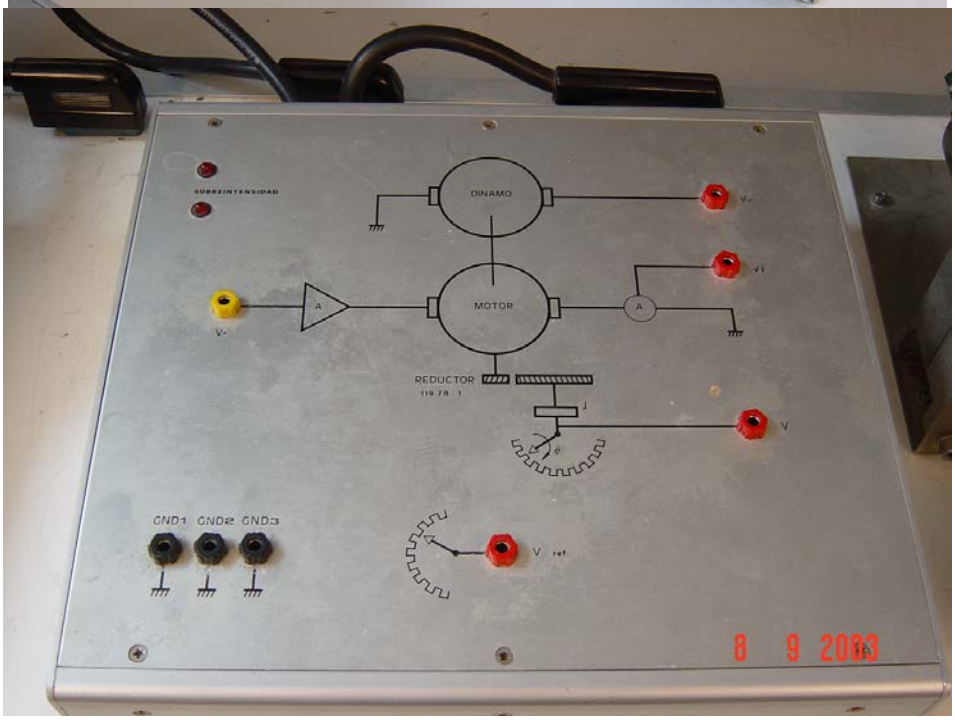
Práctica 9

Control de posición / velocidad. Implementación programada en el entorno de desarrollo CVI



OBJETIVOS

En esta práctica se diseñarán reguladores de posición y velocidad de tiempo discreto para la maqueta de accionamiento electromecánico utilizada en la práctica 4. Los controladores diseñados serán implementados en el computador por medio de aplicaciones generadas por medio del entorno de desarrollo comercial que fue utilizado por primera vez en la práctica anterior (*CVI* de *National Instruments*).



DESCRIPCIÓN DEL SISTEMA A CONTROLAR

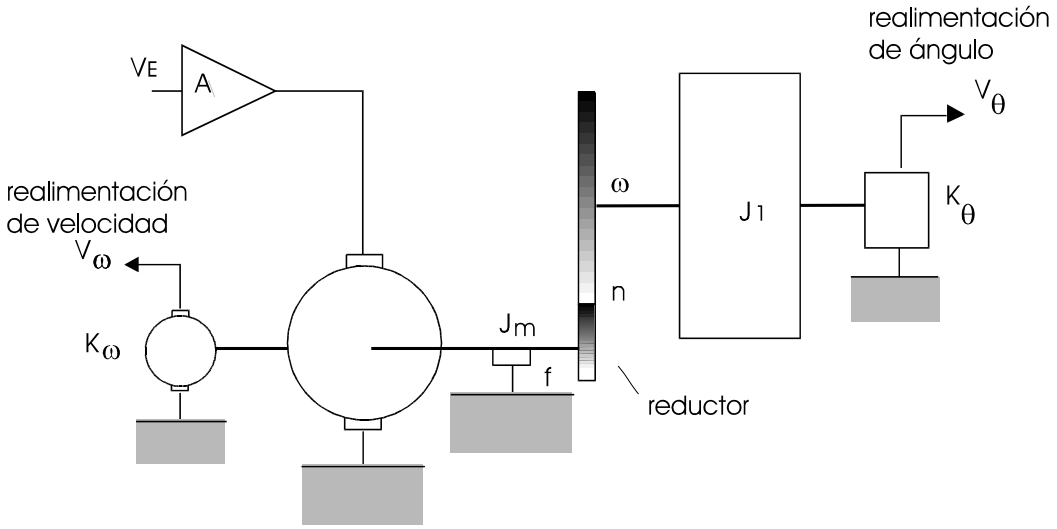


Diagrama esquemático del Servosistema.

La figura representa el esquema del accionamiento electromecánico disponible en el laboratorio. Este sistema consta de un amplificador de potencia de ganancia A, un motor de corriente continua controlado por inducido, un reductor de relación de reducción n y la inercia J1 . Sobre el eje del motor actúa una fricción viscosa de valor f . Para permitir el control de dicho sistema se dispone de dos sensores, uno que mide la posición θ del eje de la inercia de constante Kθ V/rad y una dínamo tacométrica sobre el eje del motor de constante Kω V/rad*sg^{-1} .
Existen en el motor una fricción seca y juego en el reductor que se pueden modelar como perturbaciones aditivas sobre el par del eje.

Los valores numéricos de los parámetros del sistema son:

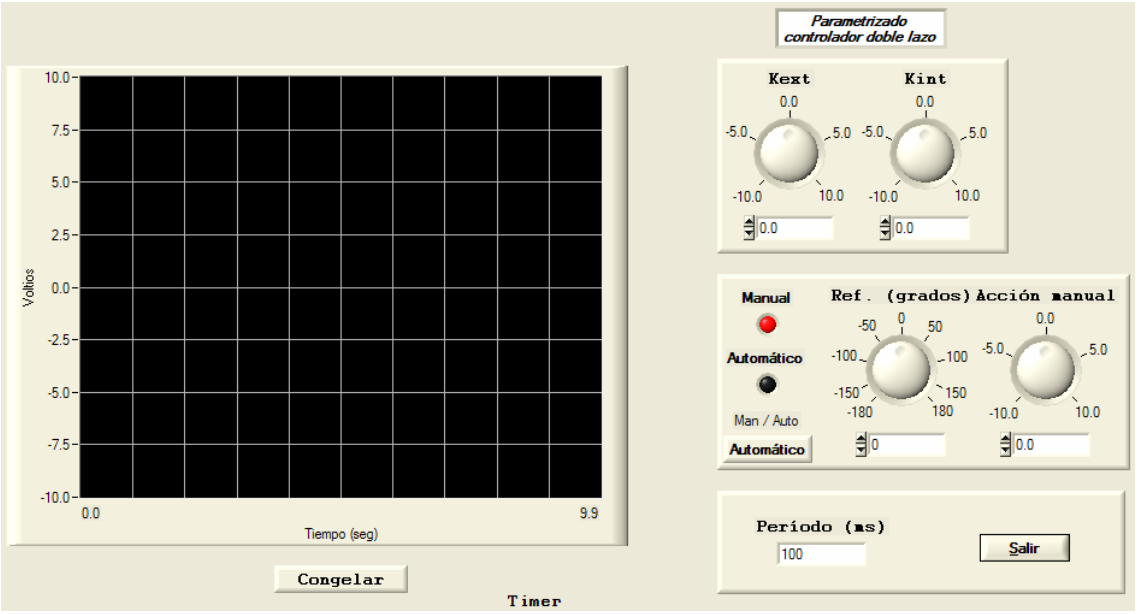
A=2’7
R=2’79Ω
Kp=0’0729Nw*m/A
Ke=0’07289 V/rad* sg^{-1}
n=119’75
f=0’655*10^{-4}Nw*m /rad* sg^{-1}
J=1’08*10^{-4} Kg*m^2 Inercia total referida al eje del motor (Jm+J1/n^2)

Sensores:
Kθ =2’75 V/rad
Kω =2’792*10^{-2}V/rad*sg^{-1} .

BREVE DESCRIPCIÓN DE LA APLICACIÓN

Edición de la GUI

La siguiente imagen refleja la GUI prediseñada para esta práctica. En la primera parte no deberá modificarse, si bien para el control de velocidad habrá que adaptarla levemente. La GUI y su correspondiente fichero de enlace están contenidas en los ficheros p10.uir y p10.h respectivamente



Software de E/S

En las aplicaciones de instrumentación y control, el software de manejo de los controladores específicos de E/S forma parte de la aplicación. En nuestro caso va a ser utilizado un bus de campo ya utilizado en prácticas anteriores: INTERBUS, de Phoenix Contact. (http://www.phoenixcontact.com). De entre sus características, cabe destacar:

- Bus determinista
- Arquitectura de 2 entradas + 2 salidas analógicas
- Precisión: 14 bits

El software de manejo reside en una DLL cuyas funciones están prototipadas en el correspondiente archivo de cabecera. Para facilitar el manejo de esta tarjeta se han creado 2 ficheros (InterBus_E_S.h y InterBus_E_S.c) exclusivamente con las funciones a utilizar (inicialización, lectura/escritura analógica y liberación de uso) Dichas funciones evitan al usuario la parametrización detallada del bus, así como ciertos detalles de la implementación de la E/S que no son merecedoras de atención en nuestro contexto. El cuadro siguiente muestra el contendido del fichero InterBus_E_S.h

```
/*********************************************************************/
/*********     INTERFAZ BASICA INTERBUS IBS PCI SC-IT  ***************/
/*********************************************************************/
/********* Autor:     ANTONIO ROMEO TELLO              ***************/
/********* Paquete:   InterBus_E_S                     ***************/
/********* Lenguaje:  ansi C                           ***************/
/********* Mision:    Proporciona herramientas para    ***************/
/*********            realizar la entradas/salida      ***************/
/*********            Analógica              .         ***************/
/********* Fecha:     ABRIL- 2005                      ***************/
/*********************************************************************/

#include "g4hliw32.h"

#ifdef __cplusplus
extern "C" {            /* Assume C declarations for C++ */
#endif  /* __cplusplus */

  extern HLIRET init_IB (void);
  /* Inicializa la tarjeta controladora de INTERBUS para el trabajo  */
  /* con el módulo de E/S analógicas en modo de disparo por software */
  /* y sincronización por encuesta.  */
  /* Debe ser llamado una vez al comienzo del uso de INTERBUS        */

  void release_IB (void);
  /* Libera el bus. Debe ser llamado una vez al fnalizar su uso      */

  double analog_input_IB (int analog_channel);
  /* canales de entrada analógicos: 1..2 */

  void analog_output_IB (double data);
  /* Un único canal de salida analógica */

#ifdef __cplusplus
} /* extern C*/
#endif
```

Implementación del código fuente de la aplicación

El código fuente de la aplicación, contenido en el fichero p10.c, fue generado (en primera instancia) de forma automática con *CodeBuilder*. Contiene, además de la atención a los eventos generados desde la GUI (*Callback functions*), el programa principal, cuya misión fundamental es la carga de la GUI y la ejecución del entorno que se encarga de generar y capturar los eventos asociados a los elementos de la misma. El cuadro siguiente muestra el programa principal con la inclusión posterior de las funciones de inicialización y liberación del Bus de campo:

```
int main (int argc, char *argv[])
{
	if (InitCVIRTE (0, argv, 0) == 0)
		return -1;	/* out of memory */
	if ((panelHandle = LoadPanel (0, "p10.uir", PANEL)) < 0)
		return -1;
	if (init_IB () == HLI_OKAY)
	{
		DisplayPanel (panelHandle);
		RunUserInterface ();
		DiscardPanel (panelHandle);
		analog_output_IB (0);
		release_IB ();
	}
	return 0;
}
```

La rutina de servicio del evento asociado al reloj

Por último, en toda aplicación periódica (en nuestro caso, las tareas de control lo son) la rutina que se ejecuta cada período de muestreo deberá formar parte de la atención al evento *Timer_tick* de un reloj *(timer)* que el usuario deberá haber incluido en la GUI. Este reloj no es un elemento visible en la fase de ejecución de la aplicación, pero permite la generación de eventos temporizados que, a la postre, permitirán desencadenar las operaciones conducentes al cálculo y posterior aplicación de la acción de control.

```
/*****************************************************************************/
/* Timer_tick: ejecuta el algoritmo de control cada vez que dispara el tick del timer */
/*    Juega un papel equivalente al de la rutina de servicio de interrupción de reloj */
/*****************************************************************************/

int CVICALLBACK Timer_tick (int panel, int control, int event,
               void *callbackData, int eventData1, int eventData2)
{
        switch (event)
               {
               case EVENT_TIMER_TICK:
                       {
                       pos=analog_input_IB (1);
                       error=ref-sal;
                       integral_error=integral_error+T*error;
                       ……
               }
        return 0;
}
```

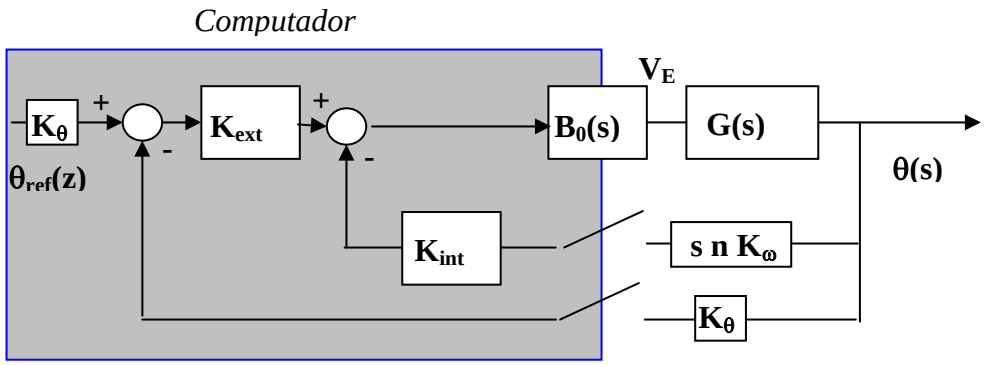
Más info.: en la guía rápida de CVI (http://www.ni.com/pdf/manuals/323552b.pdf)

1ª PARTE: CONTROL DE POSICIÓN DE DOBLE LAZO

En esta primera parte diseñaremos por los métodos expuestos en clase de teoría el servocontrol de doble lazo proporcional que permita cumplir las siguientes especificaciones:

$e_p=0$
SO=0
$t_r \leq 1$ sg

La figura representa esquemáticamente la estructura de control propuesta:



Donde la función de transferencia del sistema a controlar G(s) es:

$$G(s) = \frac{0.299}{s(1+0.055s)}$$

Estudio teórico

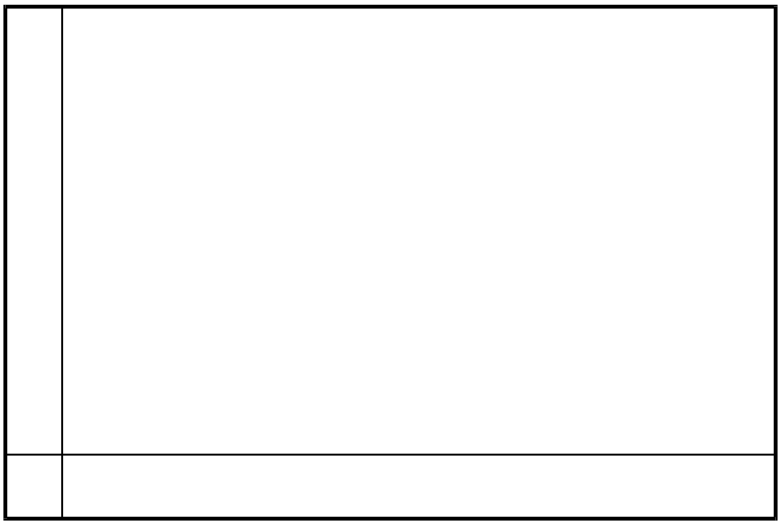
ET1-Utilizando el método de **emulación programada de reguladores continuos**, diseñar el regulador proporcional de doble lazo para cumplir las especificaciones de funcionamiento expuestas anteriormente. Elegir un valor para el período de muestreo, discretizar el regulador y extraer las ecuaciones en diferencias de control.

Estudio práctico

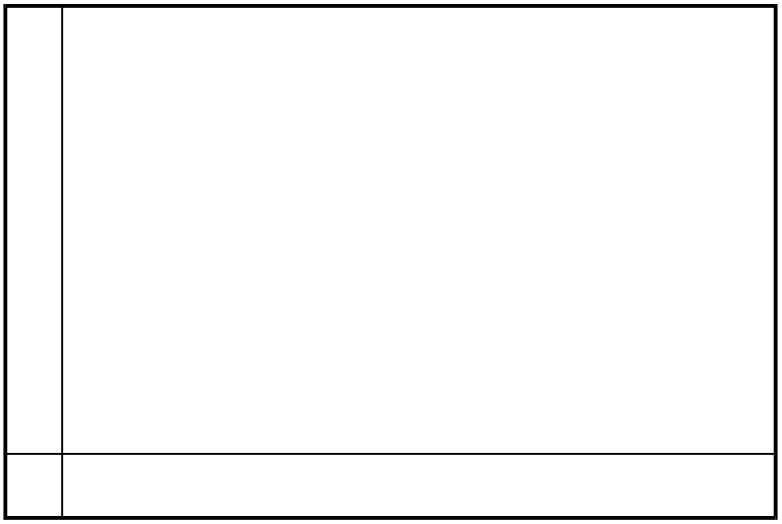
Copiad sobre el escritorio la carpeta correspondiente del servidor de prácticas departamental **\\Iis1\practicas\reg_aut_electronicos\prac10**.

Abrid el fichero de proyecto de CVI denominado **p10.prj**. Una vez abierto el proyecto deberemos añadirle (Edit ->add file to Project…) el fichero de librería **g4hliw32.lib** del software HLI de INTERBUS **c:\ibsg4hli\win32\msc**

EP1-Completar la implementación del regulador obtenido. El período de muestreo se modificará por medio de la correspondiente entrada dispuesta a tales efectos. Nótese que una modificación del mismo provocará la inicialización de la representación gráfica. Comprobar el cumplimento de las especificaciones para un período de muestreo de 0.02 segundos, y para 0.2 segundos. Para esta última, ¿cuál es la SO y el tiempo de respuesta?



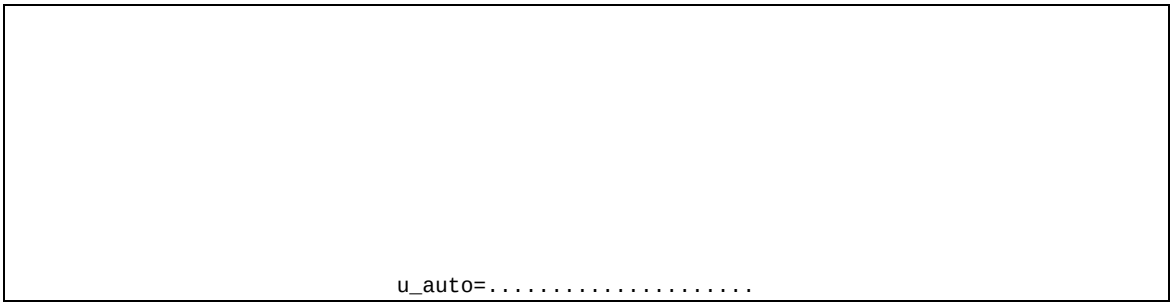
Gráfica 1 Representar la referencia, la acción y la variable controlada ante una entrada escalón unitario para T=0.02.



Gráfica 2 Representar la referencia, la acción y la variable controlada ante una entrada escalón unitario para T=0.2.

Código fuente a completar:

```
int CVICALLBACK Timer_tick (int panel, int control, int event,
              void *callbackData, int eventData1, int eventData2)
{
       switch (event)
              {
              case EVENT_TIMER_TICK:
                     pos=analog_input_IB (1);
                     vel=analog_input_IB (2);
                     if (automatico==1)
                            {
```



```
                            u_auto=.....................
                            accion=u_auto;
                            }
                     else
                            {
                            accion=u_man;
                            }
                     analog_output_9112 (accion);
                     vector_repres[0]=acomodacion*ref;
                     vector_repres[1]=pos;
                     vector_repres[2]=accion;
                     PlotStripChart (panelHandle, PANEL_GRAFICA, vector_repres, 3, 0,
                                          0, VAL_DOUBLE);
                     break;
              }
       return 0;
```

Estudio teórico

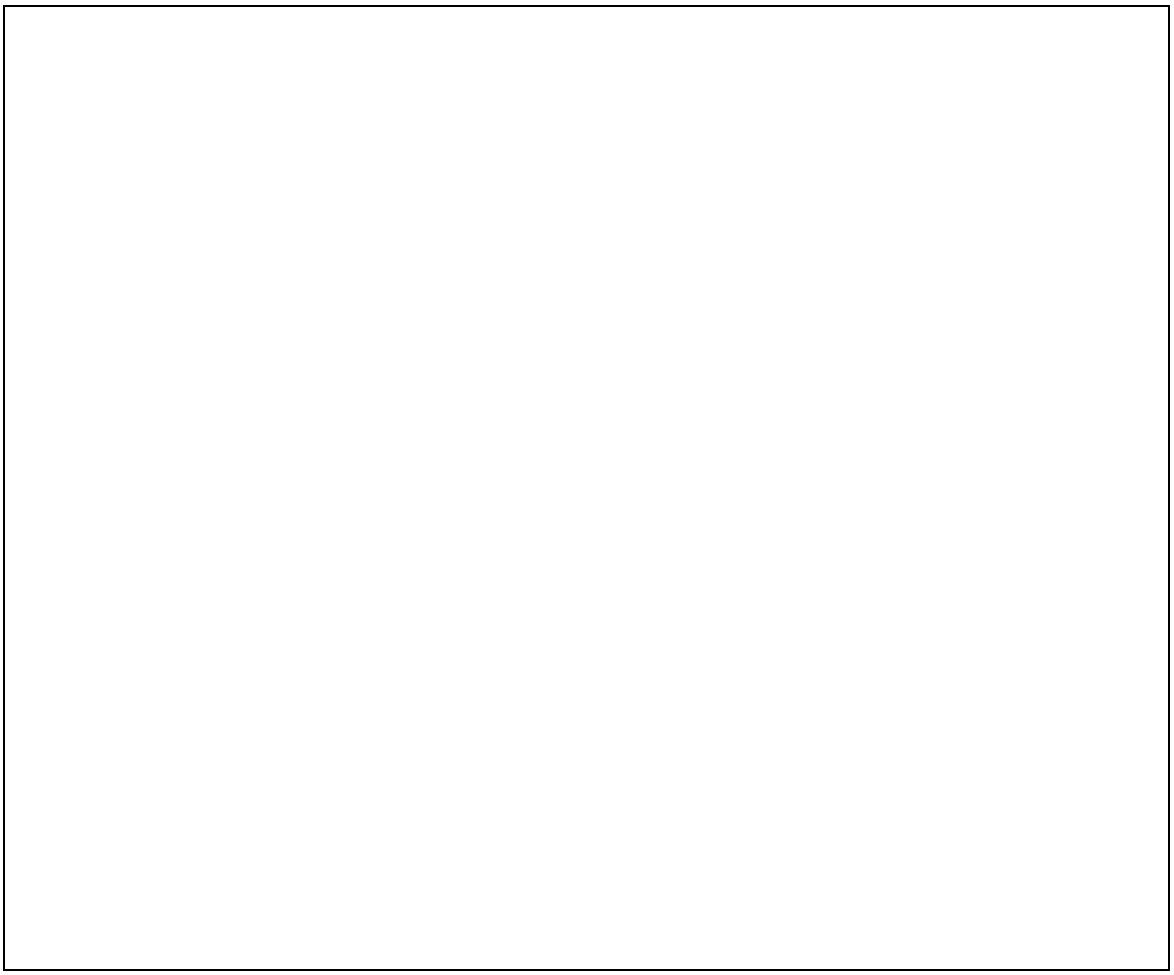
ET2- El comportamiento del sistema controlado con el regulador implementado con el período 0.2 segundos no es el estipulado en las especificaciones debido a una inadecuada elección del período de muestreo. A esta conclusión podía haberse llegado analíticamente: deduce el comportamiento en los regímenes transitorio y permanente a partir del modelo global F(z).

Como ayuda de tipo operativo, la discretización del sistema a controlar + bloqueador de orden cero (realimentación externa), obtenidas para 0.2 segundos es:

$$B_0 G(z) = \frac{0.043788(z + 0.3297)}{(z-1)(z-0.02635)}$$

Mientras que la realimentación interna discretizada se corresponde con:

$$nK_\omega B_0 Gs(z) = \frac{0.97346}{(z-0.02635)}$$

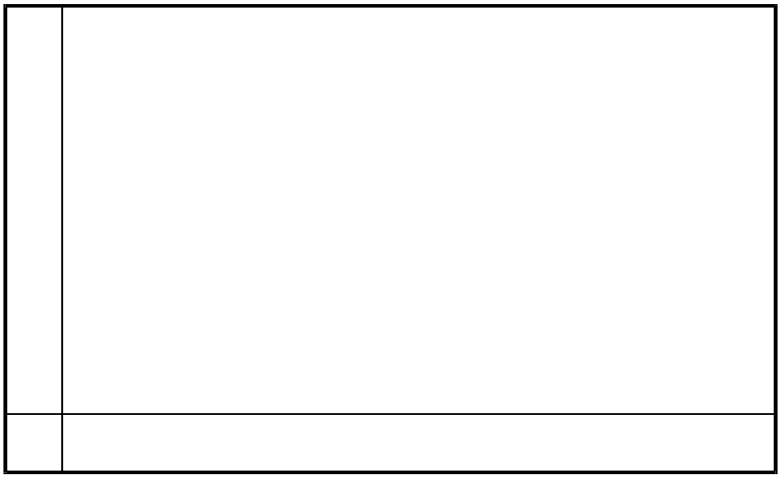


Estudio teórico

ET3- Utilizando esta vez el método de **síntesis directa en z**, diseñar el regulador proporcional de doble lazo para cumplir las especificaciones de funcionamiento expuestas anteriormente. Elegir como período de muestreo 0.2 segundos, discretizar el regulador y extraer las ecuaciones en diferencias de control.

Estudio práctico

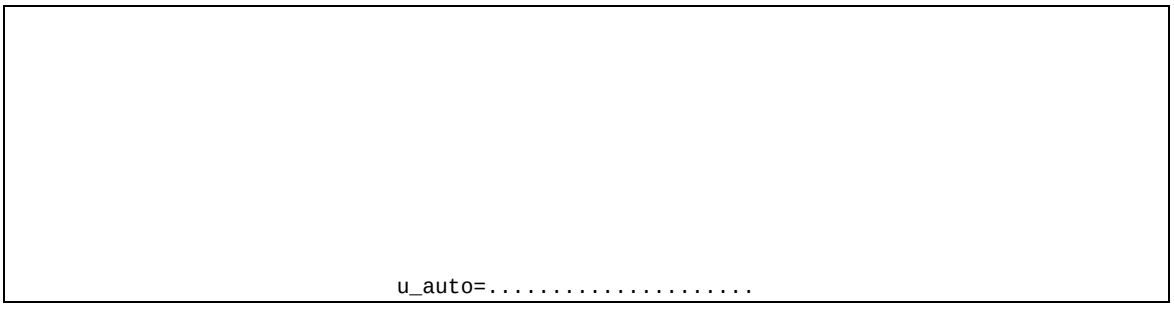
EP3-Implementar de forma programada el regulador obtenido en el estudio teórico anterior.



Gráfica 3 Representar la referencia y la variable controlada ante una entrada escalón.

Código fuente a completar

```
case EVENT_TIMER_TICK:
        pos=analog_input_IB (1);
        vel=analog_input_IB (2);
        if (automatico==1)
                {
```



```
                u_auto=.....................
                accion=u_auto;
                }
        else
                {
                accion=u_man;
                }
        analog_output_9112 (accion);
        vector_repres[0]=ref;
        vector_repres[1]=sal;
        vector_repres[2]=accion;
        PlotStripChart (panelHandle, PANEL_GRAFICA, vector_repres, 3, 0,
                                0, VAL_DOUBLE);
        break;
```

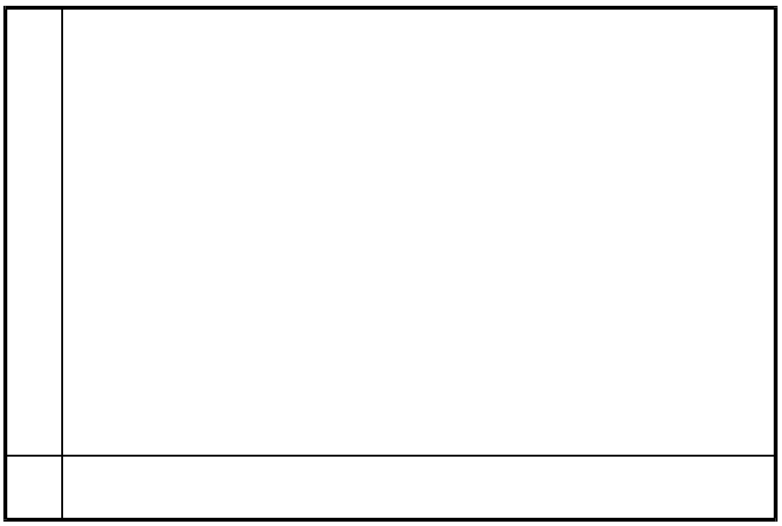
2ª PARTE: CONTROL DE VELOCIDAD

En esta segunda parte diseñaremos por los métodos expuestos en clase de teoría el servocontrol de velocidad que permita cumplir las siguientes especificaciones:

$e_p=0$
$SO=0$
$t_r \leq 0.5$ sg

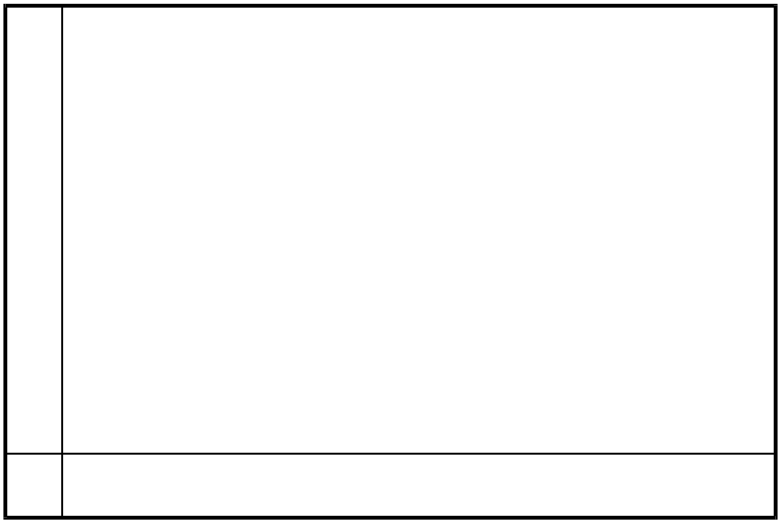
ET4- Utilizando el método de emulación programada de reguladores continuos , diseñar un regulador de manera que se consigan las especificaciones expuestas. Elegir un valor para el período de muestreo, discretizar el regulador y extraer la ecuación en diferencias de control.

EP4- Implementar de forma programada el regulador obtenido en ET4. Para ello se recomienda modificar la notación de las constantes de la GUI por otras más coherentes con la estructura del regulador obtenido. Asimismo, modificar las unidades de la referencia (rad/seg), ajustando de paso el parámetro de acomodación.

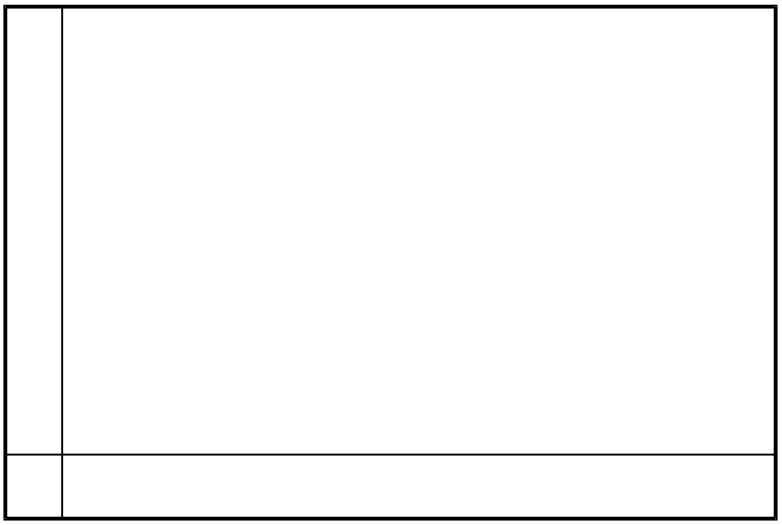


Gráfica 4 Representar la referencia y la variable controlada ante un escalón de referencia.

EP5- Llevar la referencia de velocidad a valores inalcanzables, a la vista de la saturación de nuestros actuadotes. Transcurrido un tiempo, devolver el valor de referencia a valores alcanzables. ¿Qué es lo que sucede?. Implementa un método antiwindup para evitar el crecimiento desmesurado de la integral.



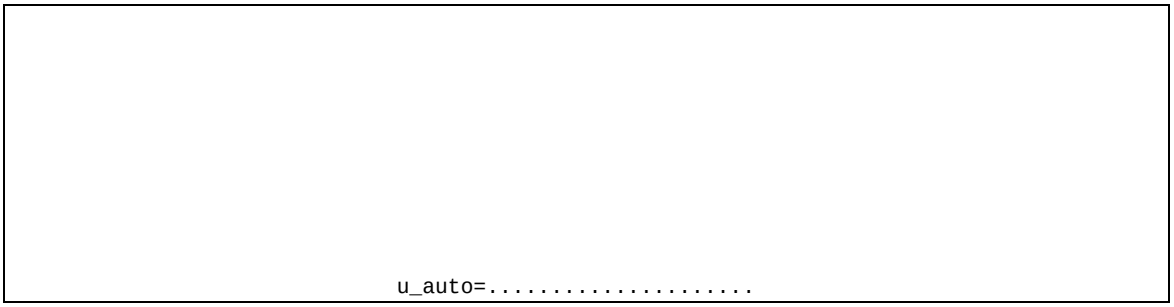
Gráfica 5 Representar el estado del sistema sin durante y después de la saturación sin método antiwindup .



Gráfica 6 Representar el estado del sistema sin durante y después de la saturación con método antiwindup implementado.

Código fuente a completar

```
            case EVENT_TIMER_TICK:
                    vel=analog_input_IB (2);
                    if (automatico==1)
                            {
```



```
                            u_auto=.....................
                            accion=u_auto;
                            }
                    else
                            {
                            accion=u_man;
                            }
                    analog_output_9112 (accion);
                    vector_repres[0]=ref;
                    vector_repres[1]=sal;
                    vector_repres[2]=accion;
                    PlotStripChart (panelHandle, PANEL_GRAFICA, vector_repres, 3, 0,
                                    0, VAL_DOUBLE);
                    break;
            }
    return 0;
```