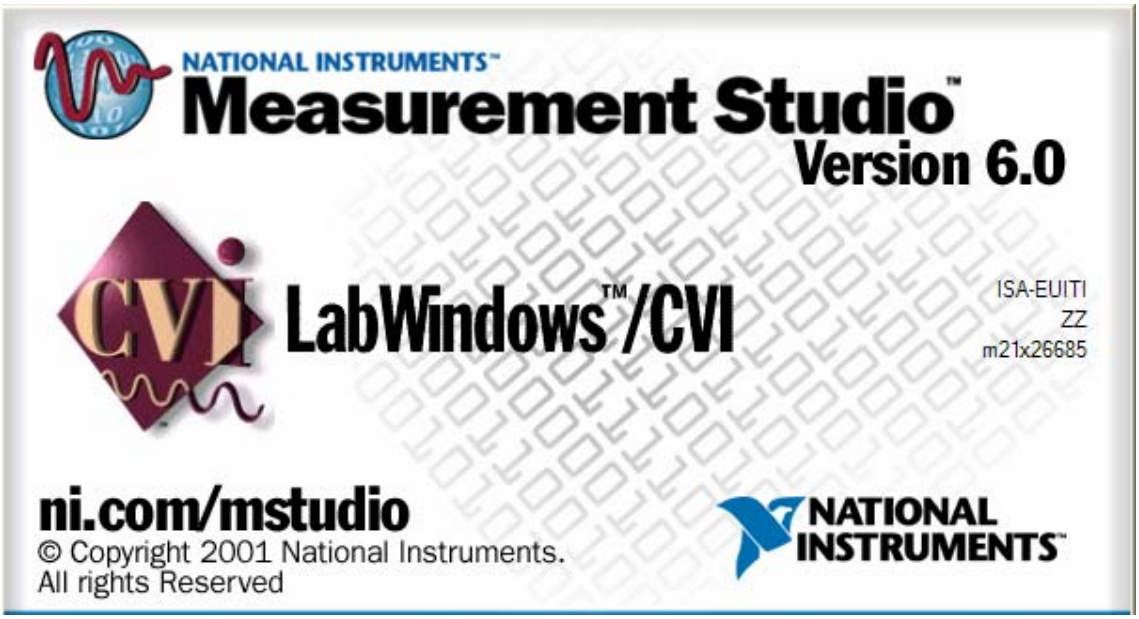
Práctica 8

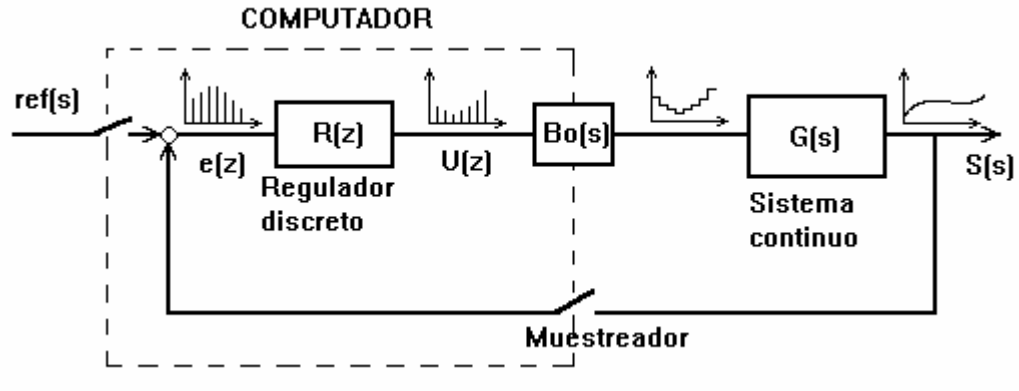
Implementación programada de reguladores. Introducción al entorno de desarrollo CVI



OBJETIVOS

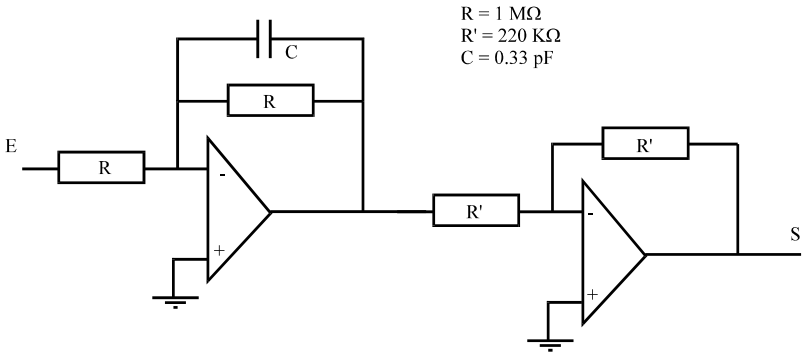
En esta práctica se estudia el comportamiento (análisis) de los sistemas contínuos controlados mediante reguladores de tiempo discreto (Algoritmo de control ejecutándose en un computador). También se diseñarán reguladores de tiempo discreto que se implementarán sobre un computador. Las aplicaciones serán desarrolladas utilizando un entorno de desarrollo comercial para esta finalidad (*CVI* de *National Instruments*). Se adjunta manual de usuario del mismo.

El sistema de la figura representa esquemáticamente el control de procesos contínuos por medio del computador:

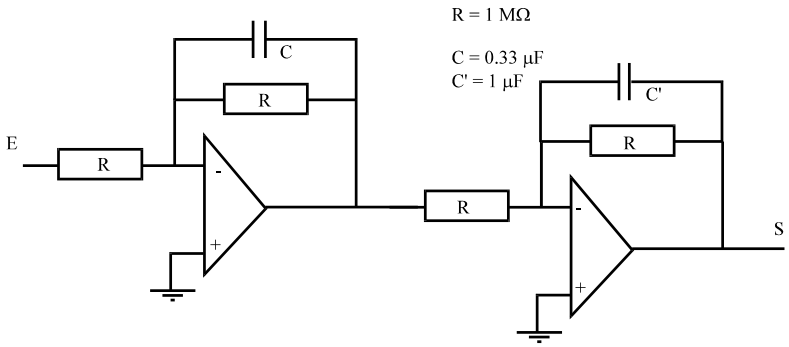


El sistema continuo a controlar está realizado por medio de circuitos basados en el amplificador operacional. Su función de transferencia puede ser de primer o segundo orden en función de que el interruptor correspondiente esté o no abierto:

$$G(s) = \frac{1}{1+0.33s} \qquad \text{o bien} \qquad G(s) = \frac{1}{(1+0.33s)(1+s)}$$



Esquema circuital del sistema de primero orden.



Esquema circuital del sistema de segundo orden.

BREVE INTRODUCCION A LA HERRAMIENTA

CVI (*C for Virtual Instrumentation*) es un entorno de desarrollo de Nati*onal Instruments*, (http://www.ni.com) específicamente pensado para la generación de aplicaciones en las que, de una forma u otra intervienen instrumentos de medida y la correspondiente interfase con el computador. En nuestro ámbito, las aplicaciones de control por computador utilizan instrumentos de medida conectados a tarjetas de adquisición de datos, por lo que CVI es susceptible de ser utilizado con el fin de diseñar de manera sencilla una aplicación de control que contendrá, al menos:

1. Un interfaz de usuario con una fuerte componente visual. Dicha interfaz contemplará todos aquellos elementos de representación gráfica o numérica, además de aquellos elementos, tanto de parametrizado de la aplicación en cualquiera de sus facetas (algoritmo de control, la propia interfaz gráfica…), como de explotación (arranque, parada, modo de operación…).
2. El algoritmo de servocontrol, que constituirá el núcleo de la aplicación
3. El manejador del dispositivo específico de E/S, proporcionado en su mayor parte por el fabricante de la tarjeta de adquisición.

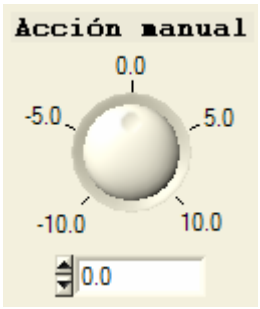
De los componentes anteriormente enumerados, la GUI conllevaría el mayor esfuerzo de programación si únicamente se contara con un lenguaje de programación de propósito general, aún cuando éste tuviese una orientación “visual”. Este motivo supone una razón de suficiente peso como para justificar la existencia de herramientas como la que nos ocupa, facilitadoras de esta etapa del diseño. El proceso de diseño de este tipo de aplicaciones comprende al menos 2 pasos:

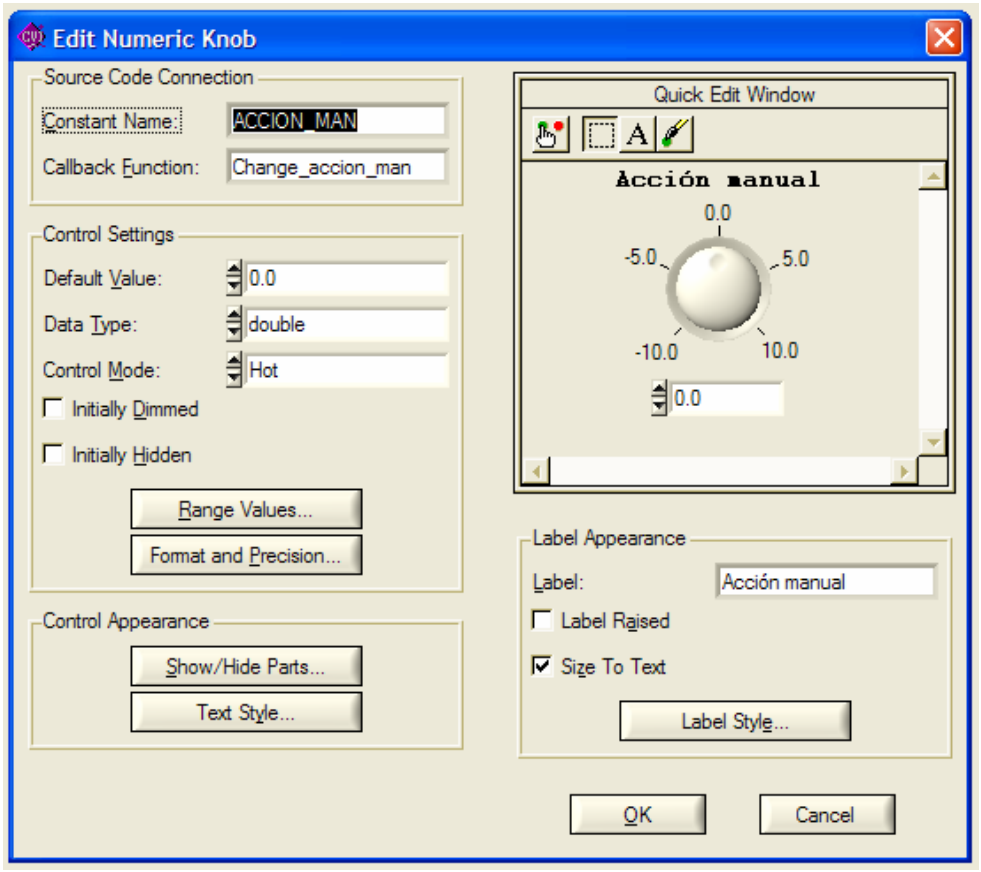
- La edición de la GUI
- La implementación del código asociado al programa principal, así como a los eventos generados desde la GUI

Además, en nuestro caso deberemos integrar en la aplicación aquellas funciones de E/S asociadas al hardware disponible. En las siguientes secciones nos ocuparemos de cada una de estas cuestiones.

Edición de la GUI

En primer lugar la creación de la GUI (Graphic User Interface) por medio de un editor específicamente pensado para tal fin. Los objetos creados con esta herramienta son referenciados en un archivo de cabecera de lenguaje C generado automáticamente, en el que para cada objeto se contempla un identificador elegido por el usuario en la fase de edición de la GUI; además en este fichero están prototipadas aquellas funciones (*Callback functions*) asociadas a la actuación del usuario sobre el elemento (eventos de ratón u otros…). Las figuras siguientes muestran respectivamente, un elemento de la GUI, el cuadro de diálogo que permite su parametrizado durante la edición y la sección del fichero de cabecera generado automáticamente en la que se muestra toda referencia al mencionado elemento.





```
/* LabWindows/CVI User Interface Resource (UIR) Include File             */
/* Copyright (c) National Instruments 2005. All Rights Reserved.         */
/*                                                                       */
/* WARNING: Do not add to, delete from, or otherwise modify the contents  */
/*          of this include file.                                        */
/*************************************************************************/

#include <userint.h>

#ifdef __cplusplus
    extern "C" {
#endif

     /* Panels and Controls: */

.......................................
#define  PANEL_ACCION_MAN                3    /* callback function: Change_accion_man */
.......................................

     /* Callback Prototypes: */

int  CVICALLBACK Change_accion_man(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
.......................................

#ifdef __cplusplus
    }
#endif
```

Para la realización de esta práctica se ha preparado una interfaz de usuario que no será necesario modificar (fichero p9.uir), así como el fichero de enlace (p9.h) que ha sido generado automáticamente al guardar el diseño.

Implementación del código. Generación automática con CodeBuilder

Una vez generada la GUI, debe llevarse a cabo la implementación del código correspondiente al programa principal y a los diferentes eventos asociados a los elementos, tanto de la GUI, como eventos asociados a relojes (*Ticks*). Esta fase del diseño puede verse facilitada desde CVI por medio de *Code Builder*: un generador de código que proporciona una primera versión del código fuente en la que se incluye un esqueleto de aplicación que contendrá, tanto las *Callback functions* como el programa principal. Al programador le resta especificar con detalle cuáles son las acciones que se generarán ante la llegada de los sucesivos eventos. El siguiente cuadro muestra, a modo de ejemplo, el fragmento de código asociado a los eventos generados al manipular el elemento que nos ha ocupado hasta ahora:

```
int CVICALLBACK Change_accion_man (int panel, int control, int event,
               void *callbackData, int eventData1, int eventData2)
{
       switch (event)
               {
               case EVENT_COMMIT:
                       GetCtrlVal (panelHandle, PANEL_ACCION_MAN, &u_man);
                       break;
               case EVENT_RIGHT_CLICK:
                       MessagePopup ("AYUDA", "Varía la acción manual utilizada");
                       break;
               }
       return 0;
}
```

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 utilizada una tarjeta de E/S multipropósito de ADLINK Technology Inc. (http://www.adlinktech.com). De entre sus características, cabe destacar:

- Bus PCI
- 16 entradas + 12 salidas analógicas
- Precisión: 12 bits
- 16 entradas + 16 salidas binarias
- 3 relojes independientes
- 3 diferentes modos de disparo
- Posibilidad de transferencia DMA
- Tasas de muestreo de hasta 110 KHz

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 (tarjeta_E_S.h y tarjeta_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 de la tarjeta, 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 tarjeta_E_S.h

```
/*****************************************************************************/
/*********       INTERFAZ BASICA TARJETA ACL-9112      *********************/
/*****************************************************************************/
/********* Autor:     ANTONIO ROMEO TELLO               *********************/
/********* Paquete:   TARJETA_E_S                       *********************/
/********* Lenguaje:  ansi C                            *********************/
/********* Mision:    Proporciona herramientas para     *********************/
/*********            realizar la entradas/salida       *********************/
/*********            Analógica                 .       *********************/
/*********            Modo disparo: por software.       *********************/
/*********            Sincronización: por encuesta.     *********************/
/********* Fecha:     ABRIL- 2005                       *********************/
/*****************************************************************************/


  void init_9112 (void);
  /* Inicializa la tarjeta para trabajar con las entradas analógicas */
  /* en modo de disparo por software y sincronización por encuesta.  */
  /* Debe ser llamado una vez al comienzo del uso de la tarjeta      */

  void release_9112 (void);
  /* Libera la tarjeta. Debe ser llamado una vez al fnalizar su uso  */

  double analog_input_9112 (int analog_channel);
  /* canales de entrada analógicos: 0..15 */

  void analog_output_9112 (double data);
```

Programa principal

El fichero generado 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 generado en primera instancia con *CodeBuilder* y modificado posteriormente con la inclusión posterior de las funciones de inicialización y liberación de la tarjeta de E/S

```
/*****************************************************************************/
/* Programa principal: carga la pantalla y ejecuta la interfaz de usuario    */
/*****************************************************************************/

int main (int argc, char *argv[])
{
        if (InitCVIRTE (0, argv, 0) == 0)
                return -1;     /* out of memory */
        if ((panelHandle = LoadPanel (0, "p9.uir", PANEL)) < 0)
                return -1;
        init_9112 ();
        DisplayPanel (panelHandle);
        RunUserInterface ();
        DiscardPanel (panelHandle);
        analog_output_9112 (0);
        release_9112 ();
        return 0;
}
```

La rutina de servicio del evento asociado al reloj

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:
                        {
                        sal=analog_input_9112 (0);
                        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)

Estudio teórico

ET1-Utilizando el método de emulación programada de reguladores continuos, diseñar el regulador proporcional para el sistema de primer orden (¡atención al interruptor!) que haga el error de posición menor que 0.2. Elegir un valor para el período de muestreo, discretizar el regulador y extraer la ecuación en diferencias de control.

ET2-Con el valor de K obtenido en el apartado anterior, calcula el valor del período de muestreo por encima del cual el sistema se inestabiliza.

Estudio práctico

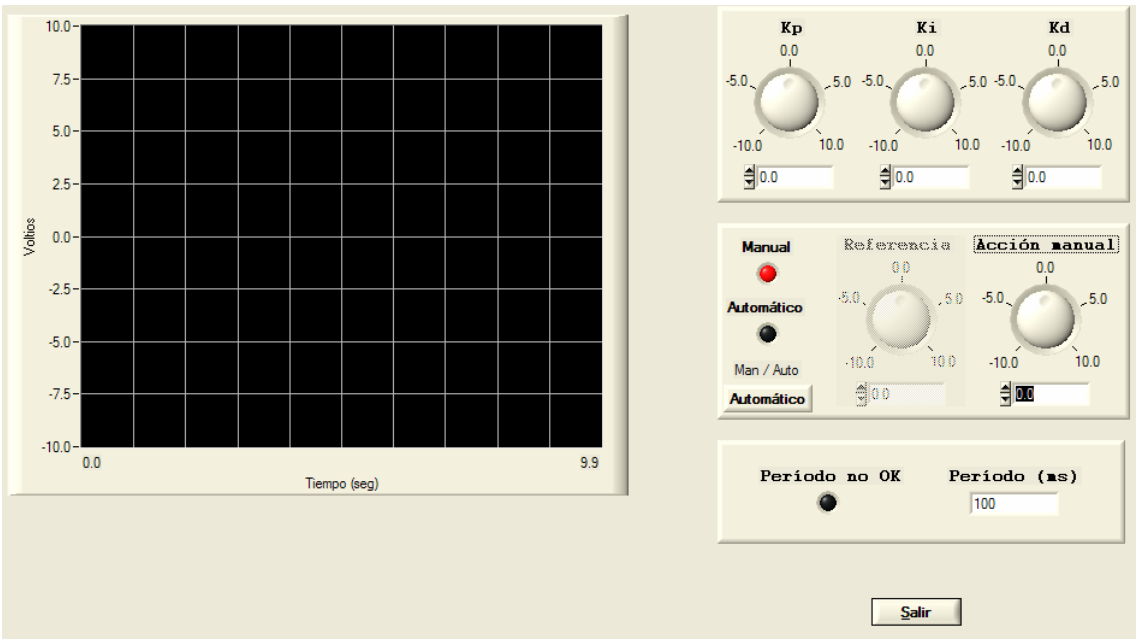
Abrid una carpeta nueva (al finalizar la práctica deberéis borrarla) en **d:\usuarios**; sobre ella copiaréis los archivos contenidos en la carpeta correspondiente del servidor de prácticas departamental **\\Iis1\practicas\reg_aut_electronicos\prac9**.

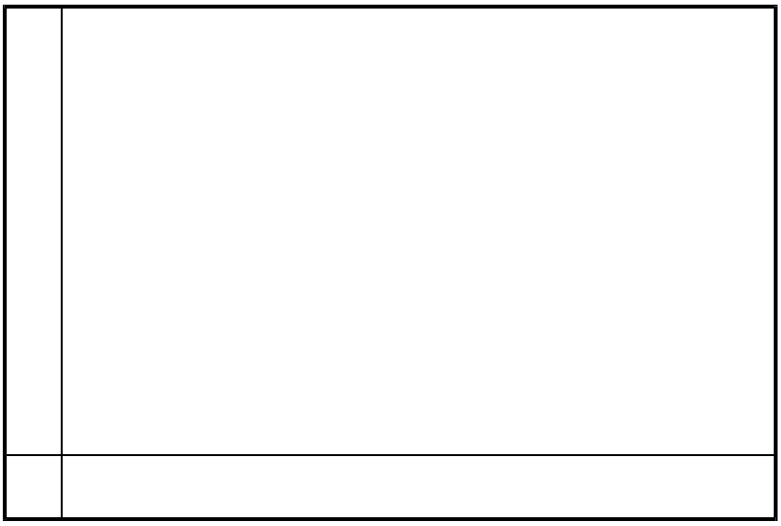
Abrid el fichero de proyecto de CVI denominado **p9.prj**. Una vez abierto el proyecto deberemos añadirle (Edit ->add file to Project…) el fichero de librería **pci_dsk.lib** del la tarjeta AdLink, que se encuentra en **c:\adlink\lib** en el que se han implementado las cuestiones referentes a la representación gráfica e inicialización.

EP1-Completar la implementación del regulador proporcional 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 la estabilidad del sistema controlado con distintos períodos de muestreo.

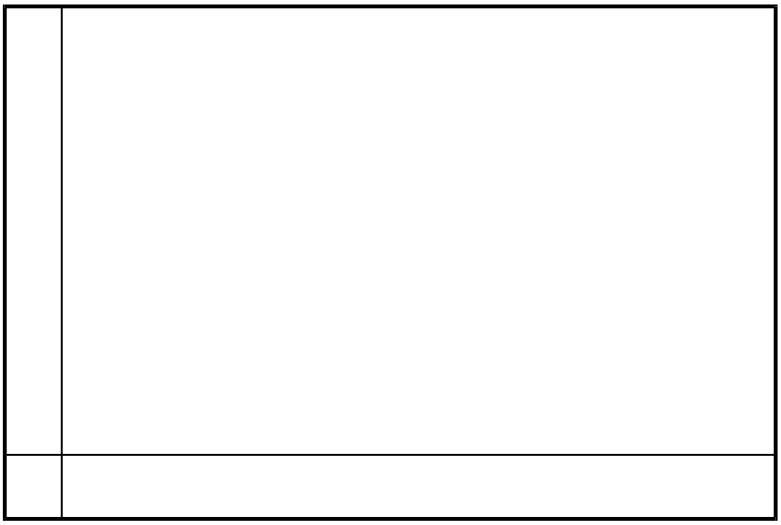
Tómese como punto de partida el fichero p9.c del proyecto que ha sido abierto con anterioridad, en el que se han implementado las cuestiones referentes a la representación gráfica e inicialización.

La GUI es la que se muestra en la figura siguiente





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



Gráfica 2 Representar la referencia, la acción y la variable controlada en el instante en que se inestabiliza debido al aumento del periodo de muestreo.

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:
                     sal=analog_input_9112 (0);
                     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;
```

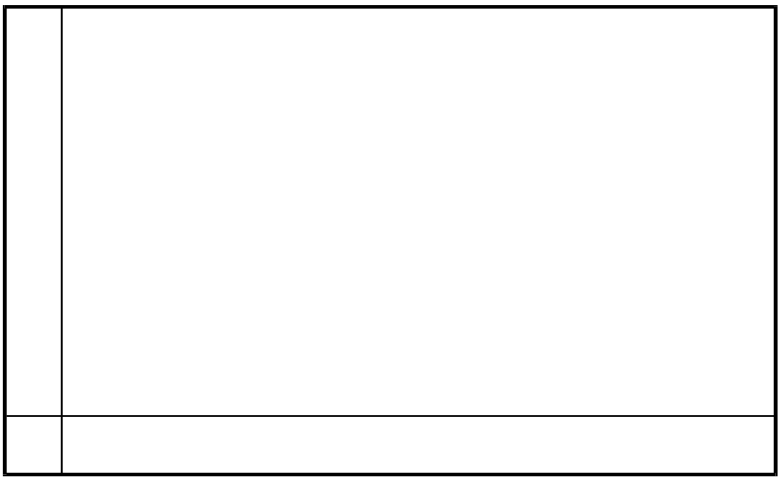
Estudio teórico

ET3- Calcular el mínimo tiempo de respuesta que se puede conseguir con un regulador integrador puro (K/s). Utiliza el mismo método de síntesis que en el caso anterior.

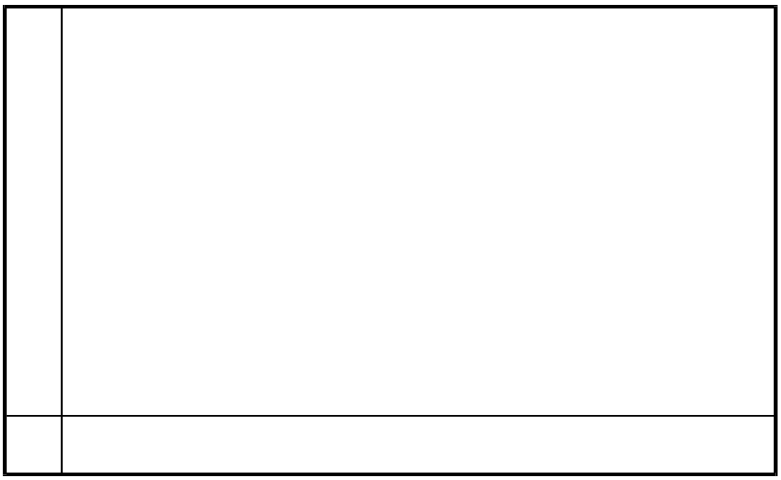
¿Qué sucede si se aumenta la ganancia del regulador?

Estudio práctico

EP3-Implementar de forma programada el regulador Integral puro obtenido en Observar la influencia de la saturación de la salida.



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



Gráfica 4 Representar la acción ante una entrada escalón.

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:
                     sal=analog_input_9112 (0);
                     if (automatico==1)
                            {
```

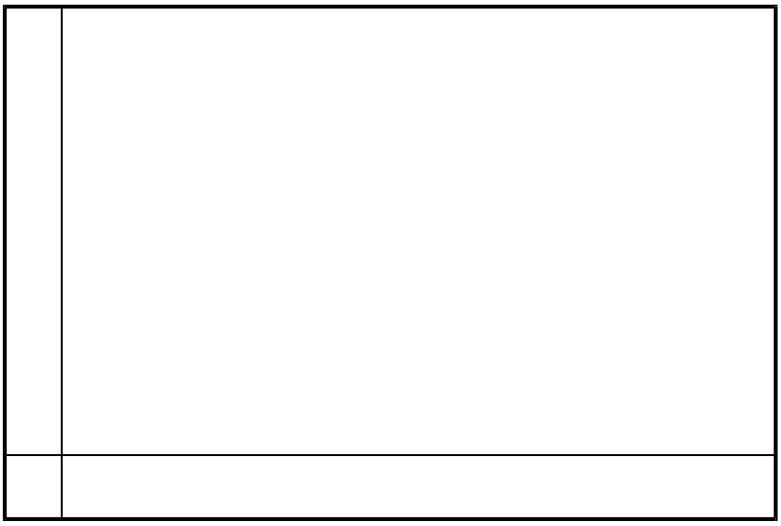
```
                        u_auto=.....................
                        accion=u_auto;
                        }
                else
                        {
                        accion=u_man;
                        }
                analog_output_9112 (accion);
                vector_repres[0]=ref;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

ET4- Utilizando el mismo método de síntesis que en los apartados anteriores, pero esta vez sobre el sistema de segundo orden (¡Atención al interruptor!), diseñar un regulador de manera que se consigan las siguientes especificaciones:

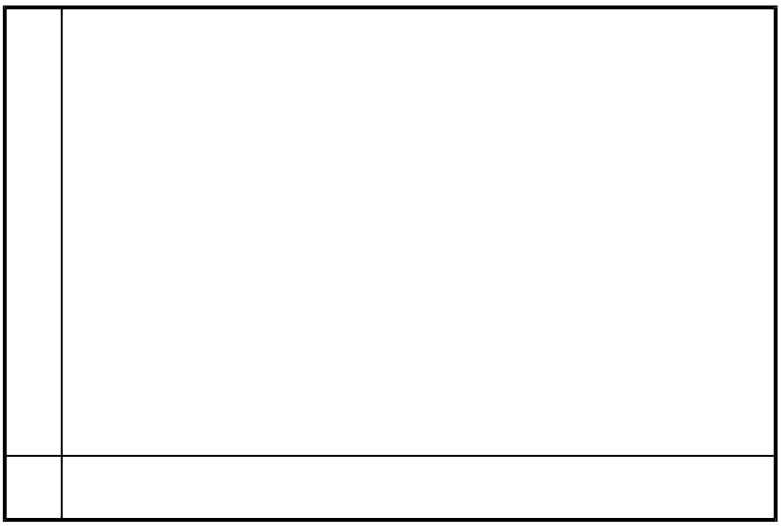
$$e_v<0.1$$
$$t_r<2.5 \text{ segundos}$$

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 Proporcional Integral obtenido en ET4. Comprobar el efecto que el paso de manual a automático tiene sobre la acción y la variable controlada. Para ello llevar el sistema en modo manual al valor de referencia (ajustarla previamente), para después conmutar al modo automático.



Gráfica 5 Representar la referencia y la variable controlada antes y después del paso a modo automático.



Gráfica 6 Representar la acción antes y después del paso a modo automático.

Código fuente a completar

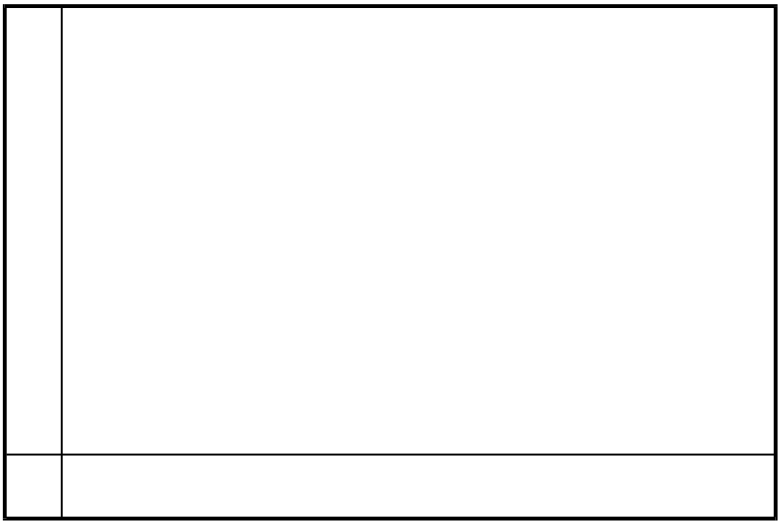
```
int CVICALLBACK Timer_tick (int panel, int control, int event,
               void *callbackData, int eventData1, int eventData2)
{
       switch (event)
               {
               case EVENT_TIMER_TICK:
                      sal=analog_input_9112 (0);
                      if (automatico==1)
                             {
```

```
                             u_auto=.....................
                             accion=u_auto;
                             }
```

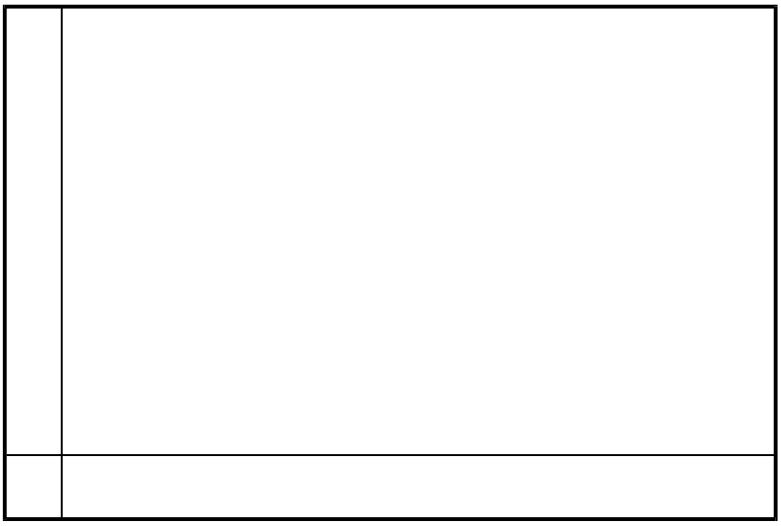
```
                else
                        {
                        accion=u_man;
                        }
                analog_output_9112 (accion);
                vector_repres[0]=ref;
                vector_repres[1]=sal;
                vector_repres[2]=accion;
                PlotStripChart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

Estudio práctico

EP5-modificar el regulador anterior de forma que se habilite el paso adecuado entre los modos manual y automático. Utilícese para ello el método de inicialización de la integral. Reproducir condiciones similares a las solicitadas en EP4 para evaluar la transferencia entre los modos manual y automático.



Gráfica 7 Representar la referencia y la variable controlada antes y después del paso a modo automático..



Gráfica 8 Representar la acción antes y después del paso a modo automático.

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:
                     sal=analog_input_9112 (0);
                     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;
```