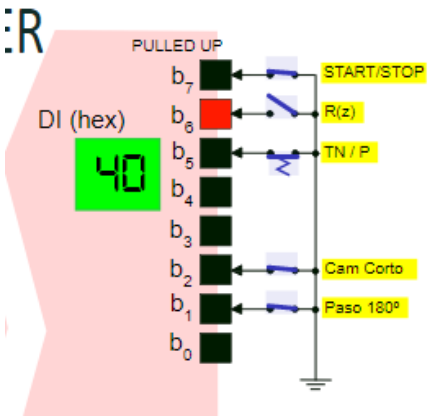
Guía de Prácticas

ASIGNATURA:	Informática Industrial y Comunicaciones		
CENTRO:	Escuela Politécnica de Ingeniería de Gijón		
ESTUDIOS:	Grado en Ingeniería Electrónica y Automática		
CURSO:	3º	CUATRIMESTRE:	1
CARÁCTER:	Obligatoria	CRÉDITOS ECTS:	6
PROFESORADO:	Ignacio Alvarez, José Mª Enguita, Borja Millán		

PRACTICA 08: Control digital en cadena cerrada

1. Se utilizará el sistema de control de motor simulado de la práctica anterior, que es necesario actualizar descargando y descomprimiendo nueva versión como se indica al final de este documento. El simulador es el mismo que para la PL7, simplemente se han añadido etiquetas a algunos pulsadores e interruptores de entrada para que el programador les dé su funcionalidad:



Realizar un programa que realice el control de posición en cadena cerrada.

La posición angular del motor debe seguir en todo momento a la consigna (potenciómetro manual de referencia), con las siguientes especificaciones:

a) Periodo de muestreo Tm=100 ms.

b) Regulador a aplicar en función de la selección por interruptores de entrada:
   - SW_7 activado: parar el motor (aplicar tensión 0V, abrir relé).
   - SW_7 no activado (cerrar relé):
     - SW_6 activado: cada pulsación de SW_5 cambia el modo de control:
       - Por defecto, control todo/nada con U=±2V
       - Control proporcional con K=0.05 V/º
     - SW_6 no activado: usar regulador $R(z) = \frac{U(z)}{E(z)} = \frac{0.15 - 0.11 z^{-1}}{1 - 0.43 z^{-1}}$

c) Se indicará en cada instante el sentido de giro del motor activando los bits de salida LED_6 (giro a derechas) o LED_7 (giro a izquierdas).

d) Se indicará en el LCD el modo de control y los valores de referencia y posición.

2) Realización PASO POR PASO:

2.a) Marco del programa: un entero indica el estado de control, y se actualiza según lo que ocurre.

```
int sw_in[2]={0,0};
int estado_control=ESTADO_POR_DEFECTO;
...
while (1)
{
   ...
   sw_in[1]= sw_in[0];
   sw_in[0]=Simulator_ReadDI();
   if (hay cambio en bit 7 de sw_in) {
      estado_control=nuevo estado según valor de bit 7 de sw_in[0];
      Aplicar cambios permanentes para nuevo estado_control:
        - bit peso 0 de valor_do para activar/desactivar relé
        - Escribir nuevo estado en LCD
   }

   // Aplicar cálculos para el estado que hay que actualizar en cada
   // instante
   switch (estado_control) {
     case ESTADO_PARADO:
       umotk_V=0;
       break;
     case ESTADO_CONTROL_POS:
       umotk_V=Calculo acción control; (por ejemplo, 2V cte.)
      break;
   }
   AplicarTensionMotor_V(umotk_V);
   Actualizar leds en valor_do
   Simulator_WriteDO(valor_do);
   Escribir refk y posk en LCD
   Retardo(TM_ms);
}
```

2.a) Añadir modos de control en una nueva variable entera: CONTROL_TN, CONTROL_P, CONTROL_RZ. Fijar el valor de dicha variable a CONTROL_TN, y realizar lazo de control todo/nada con u_{mot}=±2V

```
…
while (1)
{
   refk_deg=LeerPotRef_deg();
   posk_deg=LeerPosMotor_deg();
   errk_deg=refk_deg-posk_deg;
   ...
     case ESTADO_CONTROL:
       switch (modo_control) {
          case CONTROL_TN:
             umotk_V=ControlTodoNada(errk_deg,2.0);
             break;
       }
       break;
         ...
   AplicarTensionMotor_V(umotk_V);
   ...
}
```

2.b) Fijar vble modo_control a CONTROL_P, y añadir case en modo_control para control proporcional con Kp=0.05V

```
switch (modo_control) {
          case CONTROL_TN:
             umotk_V=ControlTodoNada(errk_deg,2.0);
             break;
          case CONTROL_P:
             umotk_V=ControlProporcional(errk_deg,0.05);
             break;
       };
```

2.c) Fijar vble modo_control a CONTROL_RZ, y añadir case en modo_control para control con una R(z) genérica. Usaremos $R(z) = \frac{U(z)}{E(z)} = \frac{0.15 - 0.11 z^{-1}}{1 - 0.43 z^{-1}}$

$$R(z) = \frac{U(z)}{E(z)} = \frac{b_0 + b_1.z^{-1} + ... + b_m.z^{-m}}{1 + a_1.z^{-1} + ... + a_n.z^{-n}}$$

$$e_k = c_k - y_k$$

$$u_k = b_0.e_k + b_1.e_{k-1} + ... + b_m.e_{k-m} - (a_1.u_{k-1} + ... + a_n.u_{k-n})$$

Valor	Simulador
Tm	100 ms
m	1
b0	0.15 V/º
b1	-0.11 V/º
n	1
a1	-0.43

Para este caso, se requieren tablas que puedan almacenar valores anteriores de ek y uk, según se ha visto en PL anteriores.

```
#define M 1
#define N 1

main()
{
   ...
   tablas tamaño M+1 para ek y b
   tablas tamaño N+1 para uk y a
   Inicializar todas las tablas
   while (1)
   {
     Desplazar tablas uk y ek
     Obtener refk_deg y posk_deg
     ek[0]=refk_deg-posk_deg;
     ...
        case CONTROL_P:
           uk[0]=...;
           break;
        case CONTROL_RZ:
           uk[0]=aplicar ec. en diferencias;
           break;
     AplicarTensionMotor_V(uk[0]);
     ...
   }
}
```

Usar función ProductoEscalar( )

2.d) Cambiar el valor de modo_control según según el estado de los interruptores y pulsadores:

```
main()
{
  ...
  int modo_control=CONTROL_TN;
  while (1)
  {
    ...
    if (hay cambio en bit 5 ó bit 6 de sw_in) {
       modo_control=nuevo modo según bits 5 y 6 de sw_in[0];
       Aplicar cambios permanentes para nuevo modo_control (LCD)
   }
    ...
   }
}
```

AMPLIACIONES:

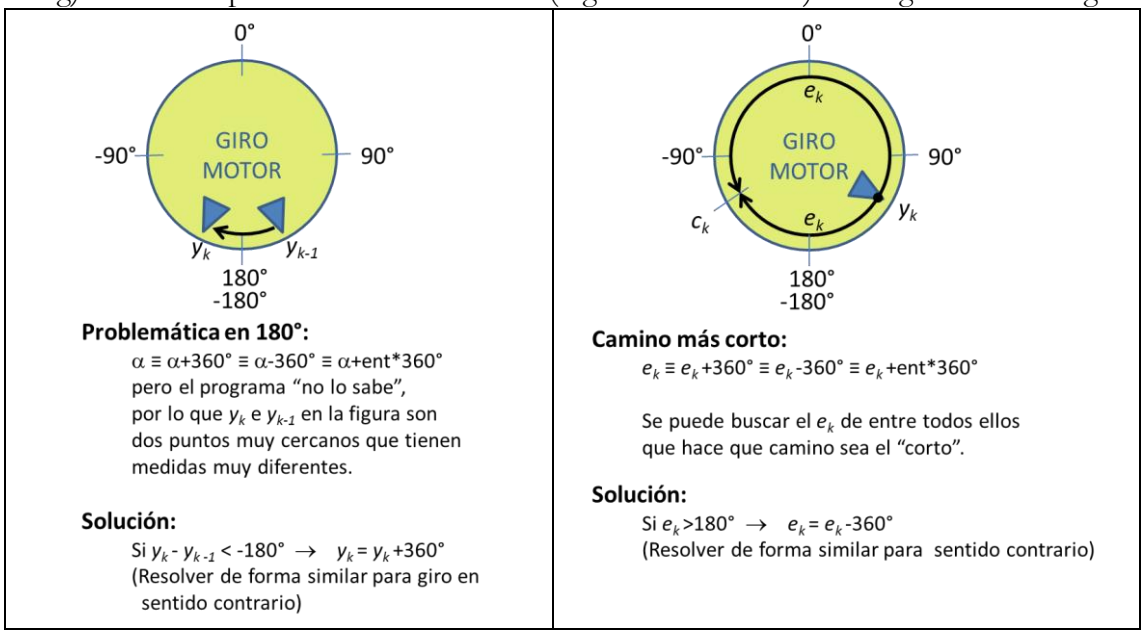
3. Añadir al cálculo de la acción de control las modificaciones necesarias para tener en cuenta el estado de los interruptores de la tabla siguiente

SW peso	Selección	Estado 1	Estado 0
1	Gestión del problema de paso por 180° (ver figura anexa)	Gestionar	No gestionar
2	Realizar el desplazamiento por el camino más corto (ver figura anexa)	Gestionar	No gestionar

3.e) Comprobar el doble problema de control en el paso de 180º:
- Cuando la referencia se sitúa cerca de 180º y la respuesta es oscilatoria.
- Cuando la referencia "salta" los 180º, no va por el camino más corto

3.f) Solucionar para el paso de 180º (según valor de SW1). Ver figura izquierda siguiente.

3.g) Solucionar para el camino más corto (según valor de SW2). Ver figura derecha siguiente.

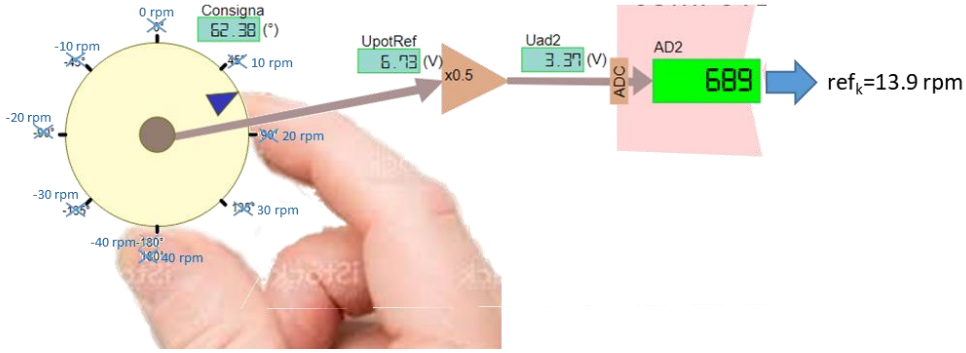


Sólo es necesario cambiar respecto al programa anterior

```
…
main()
{
   posk debe ser ahora una tabla de 2, actual y anterior
   while (1)
   {
       Desplazar Tablas Temporales (también posk)
       Leer valores más recientes de sensores
       ...
          case ESTADO_CONTROL_POS:
          {
             Si SW1 activo → saltok=posk[0]-posk[1]
                             Si saltok excesivo → actualizar posk[0]
             ek[0]=rk-posk[0];
             Si SW2 activo → Si ek[0] excesivo → actualizar ek[0]
             uk[0]=Valor Según Tipo de Control (lo mismo que antes);
          }
          Resto igual;
          ...
```

4. Usar una estructura para RZ (enteros m,n , tablas b,a), usando asignación dinámica de memoria para las tablas b y a.
5. Añadir la opción de control de velocidad, siguiendo un esquema similar: nuevo valor para `estado_control=ESTADO_CONTROL_VEL;`

Realizar pasos análogos al punto 2 pero para este nuevo estado control de velocidad, teniendo en cuenta que el selector manual ahora indica una consigna de velocidad:



ad2=Leer canal AD 2 → calc u_{ad2}(V) → calc u_{sens2}(V) → calc vel_ref (rpm)

4.d) Control todo/nada: ojo, se requiere integrador → $u_{mot} = u_{mot\ anterior} \pm 0.5V$

4.e) Control P, debe ser PI → $u_{mot} = u_{mot\ anterior} + Kp * error$ (usar Kp=0.005)

4.f) Control con R(z) genérica, usando: $R(z) = \frac{U(z)}{E(z)} = \frac{4.25 - 8.075z^{-1} + 3.866z^{-2}}{1\text{-}1.91z^{-1} + 1.16z^{-2} - 0.25z^{-3}}$

Comprobar que en todos los casos el efecto del freno es compensado por el regulador (sólo para velocidades consigna bajas, si es demasiado alta no se puede aplicar tensión suficiente).

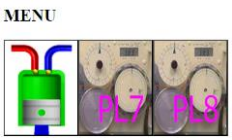
$$R(z) = \frac{U(z)}{E(z)} = \frac{b_0 + b_1.z^{-1} + ... + b_m.z^{-m}}{1 + a_1.z^{-1} + ... + a_n.z^{-n}}$$

$$u_k = b_0.e_k + b_1.e_{k-1} + ... + b_m.e_{k-m} - \\ - (a_1.u_{k-1} + ... + a_n.u_{k-n})$$

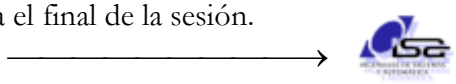
Valor	Simulador
Tm	100 ms
m	2
b_0	4.25 V/rpm
b_1	-8.075 V/rpm
b_2	3.866 V/rpm
n	3
a_1	-1.91
a_2	1.16
a_3	-0.25

SIMULADOR FEEDBACK

El sistema físico y la adquisición A/D se encuentran simulados mediante la ampliación al programa RunServer.bat. Este programa permite la visualización del estado del sistema Feedback mediante una página web en la dirección http://127.0.0.1:8080 (usuario alumno, clave ISAUNIOVI). Aparecerán las opciones de varios simuladores seleccionables mediante botones (Pistón y Feedback PL7-PL8):

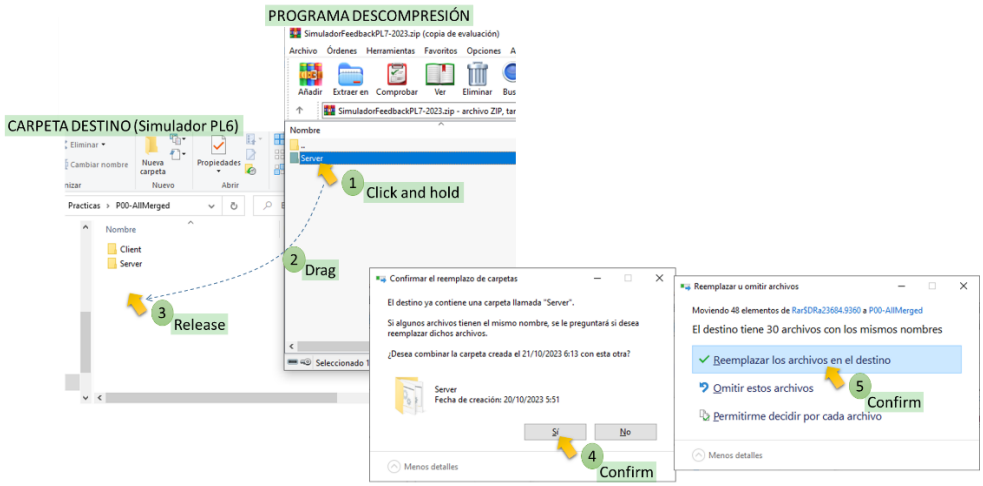


Una vez seleccionada, la visualización aparecerá la pantalla de simulación en el navegador. No se debe cerrar la aplicación en ejecución (MongooseWebServer.exe) que debe permanecer minimizada en bandeja del sistema hasta el final de la sesión.



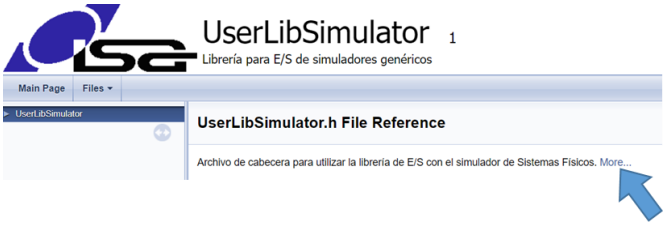
Las adiciones al programa de simulación (ya descargado en la PL anterior) se encuentran en el enlace siguiente: http://isa.uniovi.es/~ialvarez/Curso/descargas/SimuladorFeedbackPL8-2023.zip

El archivo descargado debe ser descomprimido en la misma carpeta que el simulador de las PL anteriores (asegurarse previamente que el servidor está parado con StopServer). El programa de descompresión avisará en varias ocasiones de que se van a mezclar carpetas y sustituir archivos. Se debe contestar afirmativamente a todos los avisos.



Tras la descompresión, se dispondrá de las adiciones y modificaciones necesarias en la carpeta Server.

- Server: en esta carpeta se encuentra el ejecutable RunServer.bat, que lanza el simulador (y servirá también para otros simuladores posteriores). El simulador se queda minimizado en la bandeja del sistema (abajo a la derecha, junto a los iconos de red, batería, dispositivos, etc.), y se detiene con StopServer.bat. Se debe dejar el servidor ejecutando durante toda la sesión, pero hay que **recordar detenerlo al final** para que no quede consumiendo recursos (memoria, CPU, batería).
- La carpeta Client/Library permanecerá inalterada. Se recuerda que en dicha carpeta se encuentra el archivo de ayuda help.html para el uso de la librería. Si se pulsa sobre More... en la pantalla de inicio de la ayuda, se observan los requerimientos para la compilación (archivo .pro) y un ejemplo de uso:



Para conectar a este simulador, el programador debe llamar a la función Simulator_ConnectWss( ) con los argumentos siguientes:

```
Simulator_ConnectWss("Feedback","alumno","ISAUNIOVI","127.0.0.1",8080);
```