UNIVERSIDAD DE OVIEDO

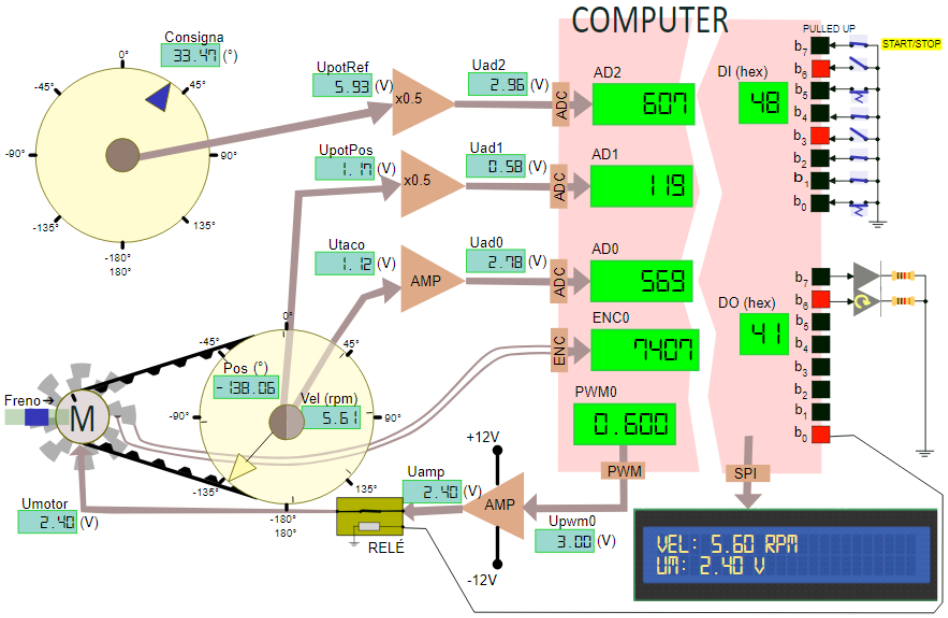
Departamento de Ingeniería Eléctrica,
Electrónica, de Computadores y Sistemas

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 07: Control básico y E/S digital

1. Ejercicio a realizar. Se dispone de un computador conectado mediante E/S analógica y digital a un sistema formado por:
   - Un motor DC de 12V, que mueve mediante una correa dentada (con relación 1:8) un disco graduado que dispone de sensores para medida de la posición y velocidad angulares.
   - Un potenciómetro para generar señal de consigna (referencia), conectado a +10V.
   - Un computador de control, con entradas A/D, salida PWM, E/S digital, y display LCD.



Se desea realizar sobre este sistema un control básico de velocidad en cadena abierta, con las siguientes especificaciones:

a) Periodo de muestreo Tm=100 ms.

b) Tensión a aplicar al motor, según entrada del usuario (-12 a 12 V), por señal PWM.

c) Activación de relé para accionamiento del motor mediante bit de peso 0 de salida DO.

d) Se indicará en cada instante el valor de las magnitudes medidas en el display LCD: posición consigna, posición y velocidad del disco conducido por el motor

e) Se indicará en cada instante el sentido de giro del disco conducido por el motor activando los bits de peso 6 de salida DO (giro a derechas) y 7 (giro a izquierdas).

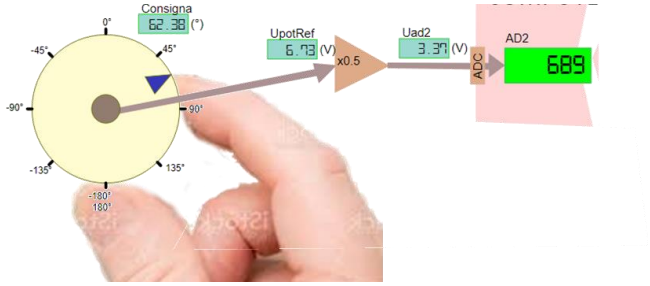
2. Para la realización de la práctica será necesario utilizar el simulador de sistema Feedback, cuya documentación para uso y descarga se encuentra al final de este documento. En este simulador, se dispone de:

   - Un motor DC (**M**) con entrada ±12VDC, y comportamiento dinámico, que mueve un disco graduado mediante una correa de relación 1:8. Al motor se le puede aplicar un freno, que simula una carga, moviendo el rectángulo **Freno** hacia el motor. El disco graduado dispone de sensores de posición (potenciómetro 0V para -180º, 10V para +180º) y velocidad (dinamo tacométrica de 200mV/rpm).
   - Un potenciómetro de consigna accionable manualmente, que entrega 0V para -180º, 10V para +180º.
   - La E/S simulada de un computador conectada al sistema anterior:
     - Tres canales A/D de 10 bits, a los cuales se conectan los sensores de posición de potenciómetro consigna (AD2), de velocidad del disco conducido por el motor (AD1) y de posición de dicho disco (AD0), a través de sendas cadenas de adaptación de señal.
     - Un encóder en cuadratura de 64x4 pulsos por vuelta, acoplado al eje del motor.
     - Una salida PWM 0-5V que gobierna la tensión aplicada al motor a través de un amplificador electrónico (0V entrada → -12V salida , 5V entrada → +12V salida).
     - Un display LCD de 2x24 caracteres.
     - Dos puertos de E/S digital de 8 bits: el puerto DI conectado a interruptores accionables manualmente (permanentes y pulsadores), el puerto DO conectado a salidas LED y a un relé para activación del moto.

   A ese simulador se puede conectar un programa de usuario mediante la librería UserLibSimulator ya utilizada en la PL anterior (Pistón).

3. **Pasos en la realización del código** (realizar y probar cada uno de forma incremental):

3.a) Obtener del canal AD 2 el valor en grados del potenciómetro manual de consigna, según la interpretación de la figura siguiente, y escribir el valor en el display LCD cada 100 ms. Posicionando el cursor sobre cada elemento de la cadena de medición se muestran las transformaciones de cada etapa, que deben ser revertidas por el cálculo.



Función:
float LeeRef( );

ad2=Leer canal AD 2 → calc u_{ad2}(V) → calc u_{sens2}(V) → calc pos_ref (deg)

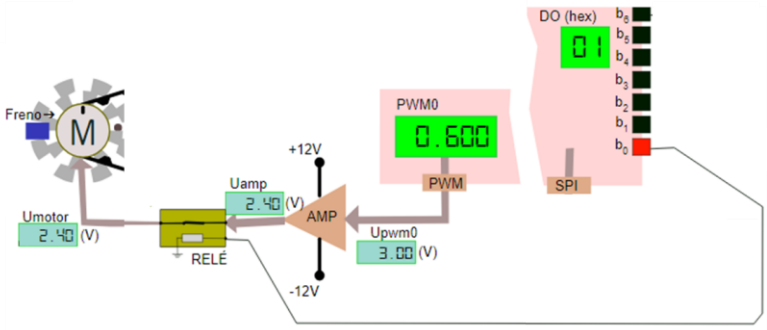
```
int err=Simulator_ConnectWss("Feedback","alumno","ISAUNIOVI",
                             "127.0.0.1",8080);
if (err!=CONNECT_IS_OK)
    return -1;
Simulator_LCD_init();

while (1)
{
   refk_deg=LeerPotRef_deg();  // Función a realizar
   Simulator_LCD_gotoxy(1,1);  // Posiciona cursor en display
   Simulator_LCD_printf(...);  // Escribe dato en display
   Simulator_Delay(100);
}
```

3.b) Activar el bit de peso 0 de las salidas digitales, de forma que se cierre el relé que permite el paso de la tensión del amplificador hacia el motor:

```
int err=Simulator_ConnectWss(...);
...
Simulator_WriteDO(valor que activa el bit de peso 0 de DO);
while (1)
{
   ...
}
```

3.c) Pedir por teclado el valor de tensión u_{mot} a aplicar al motor (-12 a 12V), y generar la salida PWM necesaria para que se aplique dicha tensión. Posicionando el cursor sobre cada elemento de la cadena de accionamiento se muestra la conversión que realiza.



Función:
void Aplica(float u_m);

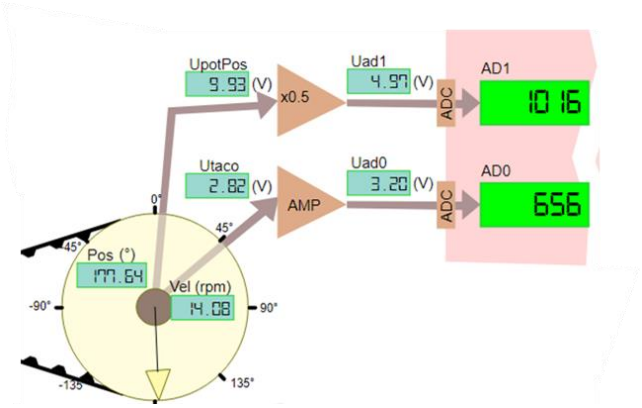
U_{mot}(V) → calc duty (tanto por 1) → calc Ton (Tpwm=cte=1000) → Aplicar PWM

```
float u_motor;
...
Simulator_ConfigPWM(0,1000,500,1);
Simulator_WriteDO(valor que activa el bit de peso 0 de DO);
pedir u_motor por teclado
AplicarTensionMotor_V(u_motor);

while (1)
   ...
```

(Comprobar que, al aplicar el freno, con la misma tensión decae la velocidad)

3.d) Modificar el anterior para la lectura cada 100 ms de la posición angular del motor (en grados) y velocidad angular (en r.p.m.), y escribir los mismos en el display LCD.



ad1=Leer canal AD1 → calc u_{ad1}(V) → calc u_{sens1}(V) → calc pos_{mot} (deg)

ad0=Leer canal AD0 → calc u_{ad0}(V) → calc u_{sens0}(V) → calc vel_{mot} (rpm)

Funciones:
float LeePos( );
float LeeVel( );

```
...
Pedir valor por teclado: umotor_V
AplicarTensionMotor V(umotor V);
while (1)
{
   refk_deg=LeerPotRef_deg();
   posk_deg=LeerPosMotor_deg();
   velk_rpm=LeerVelMotor_rpm();

   escribir datos en display
   Simulator Delay(100);
}
```

3.e) Añadir la comprobación del estado del bit 7 de pulsadores (leer DI) y accionar el relé en cada instante en función de su estado:

```
int valor_DI,valor_DO;
...
while (1)
{
   valor_DI=Simulator_ReadDI();
   if (bit de peso 7 de valor_DI está activo)
      valor_DO= valor que activa el bit de peso 0 de DO;
   else
      valor_DO= valor que desactiva el bit de peso 0 de DO;
   Simulator_WriteDO(valor_DO);
   ...
}
```

3.f) Añadir la activación de los bits 6 ó 7 de los LEDs en función del sentido de giro, modificando el valor del puerto de salida DO:

```
...
while (1)
{
   ...
   if (girando a derechas)
      valor_DO = valor deseado (LED7=0, LED6=1);
   else if (girando a izquierdas)
      valor_DO = valor deseado (LED7=1, LED6=0);
   else // Parado
      valor_DO = valor deseado (LED7=0, LED6=0);
   Simulator_WriteDO(valor_DO);
   ...
}
```

4. Ampliaciones propuestas:

   4.a) Mejorar la activación y desactivación de los LEDs de 3.e y 3.f para que la modificación de `valor_DO` no afecte al estado del resto de bits:

   Ejemplo para activar bit 7:

valor_DO	B7	B6	B5	B4	B3	B2	B1	B0
¿OP?								
¿mask?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?
result	**1**	B6	B5	B4	B3	B2	B1	B0

   Para comprobar, activar otros LEDs al principio y comprobar que se mantienen activos:

```
…
valor_DO = valor para activar LED4 y LED5;
while (1)
{

   ...
   if (giro a derechas)
      valor_DO = valor deseado (LED7=0, LED6=1, resto igual);
   else if (giro a izquierdas)
      valor_DO = valor deseado (LED7=1, LED6=0, resto igual)
   else // Parado
      valor_DO = valor deseado (LED7=0, LED6=0, resto igual);
   Simulator_Delay (100);
}
```

   4.b) Cambiar la entrada para arrancar/parar motor al bit DI5, conectado a un pulsador, de manera que cada pulsación modifique el estado anterior del motor (primera pulsación arranca, segunda pulsación para, tercera pulsación vuelve a arrancar, …)

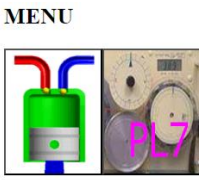
```
int valor_DI[2]; // Estados actual y anterior de pulsadores
int estado_motor[2]; // Estados actual y anterior de activación motor
…

estado_motor[0]=0;
valor_DI[0]= Simulator_ReadDI();
while (1)
{
   valor_DI[1]= valor_DI[0];
   estado_motor[1]=estado_motor[0];
   valor_DI[0]= Simulator_ReadDI();

   if (hay cambio 0 a 1 en B5 de valor_DI)
   {
      estado_motor[0]=!estado_motor[1];
      valor_DO= valor que activa bit de peso 0 de DO según estado_motor[0]
   }
   ...
   Simulator_Delay (100);
}
```

SIMULADOR FEEDBACK

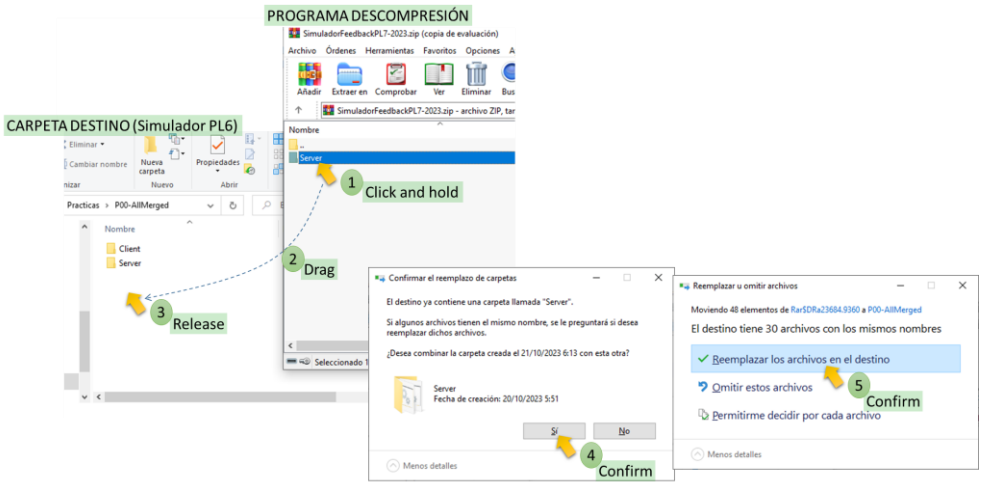
El sistema físico y la adquisición A/D se encuentran simulados mediante la ampliación al programa RunServer.bat. Este programa permite la visualización del estado del sistema Feedback mediante una página web en la dirección http://127.0.0.1:8080 (usuario alumno, clave ISAUNIOVI). Aparecerán las opciones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



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.

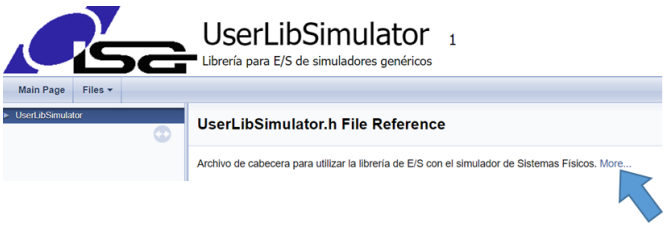
Las adiciones al programa de simulación (ya descargado en la PL anterior) se encuentran en el enlace siguiente: http://isa.uniovi.es/~ialvarez/Curso/descargas/SimuladorFeedbackPL7-2023.zip

El archivo descargado debe ser descomprimido en la misma carpeta que el simulador del pistón de la PL6 (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);
```