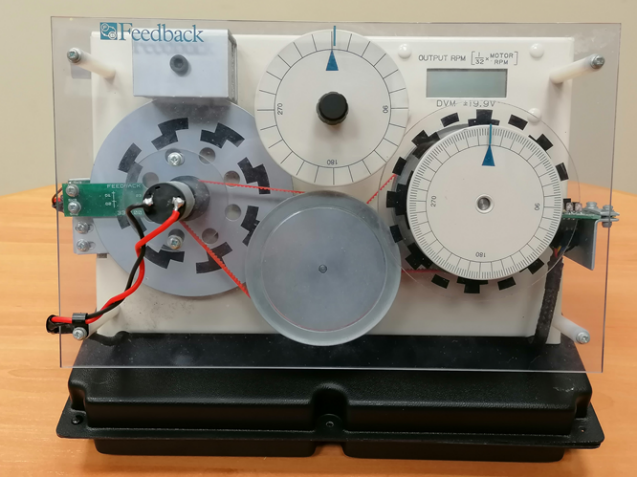
**Manual del Software Control-Feedback**

 Imagen que contiene Calendario

Descripción generada automáticamente

Contenido

[1 Software Control-Feedback 2](#_Toc100053392)

[1.1 Instalación 2](#_Toc100053393)

[1.1.1 Instalación del software en el Arduino 2](#_Toc100053394)

[1.1.2 Instalación del programa Control-Feedback en un PC 2](#_Toc100053395)

[1.2 Guía de usuario 3](#_Toc100053396)

[1.3 Modo de simulación 5](#_Toc100053397)

[1.4 Controles avanzados 6](#_Toc100053398)

[2 Programa instalado en el Arduino: controlArduino.ino 7](#_Toc100053399)

[3 Protocolo de comunicación 20](#_Toc100053400)

[4 Apéndice: Otros programas para el Arduino 21](#_Toc100053401)

[4.1 controlPlantilla 21](#_Toc100053402)

[4.2 controlArduinoComandosBasicos 25](#_Toc100053403)

[5 Apéndice: Programas en MATLAB para calibración 29](#_Toc100053404)

[5.1 identificaFB 29](#_Toc100053405)

[5.2 identifica 33](#_Toc100053406)

[5.3 tunningFB 38](#_Toc100053407)

[5.4 medidas 42](#_Toc100053408)

[5.5 comandoFB 45](#_Toc100053409)

# Software Control-Feedback

## Instalación

Para utilizar el software de control de los equipos Fedback es necesario instalar en el microcontrolador Arduino el programa de control “controlArduino.ino” y en el PC que se va a comunicar con él por el puerto USB, los drivers del puerto USB para el Arduino y la carpeta portable con el programa “Control-Feedback”.

### Instalación del software en el Arduino

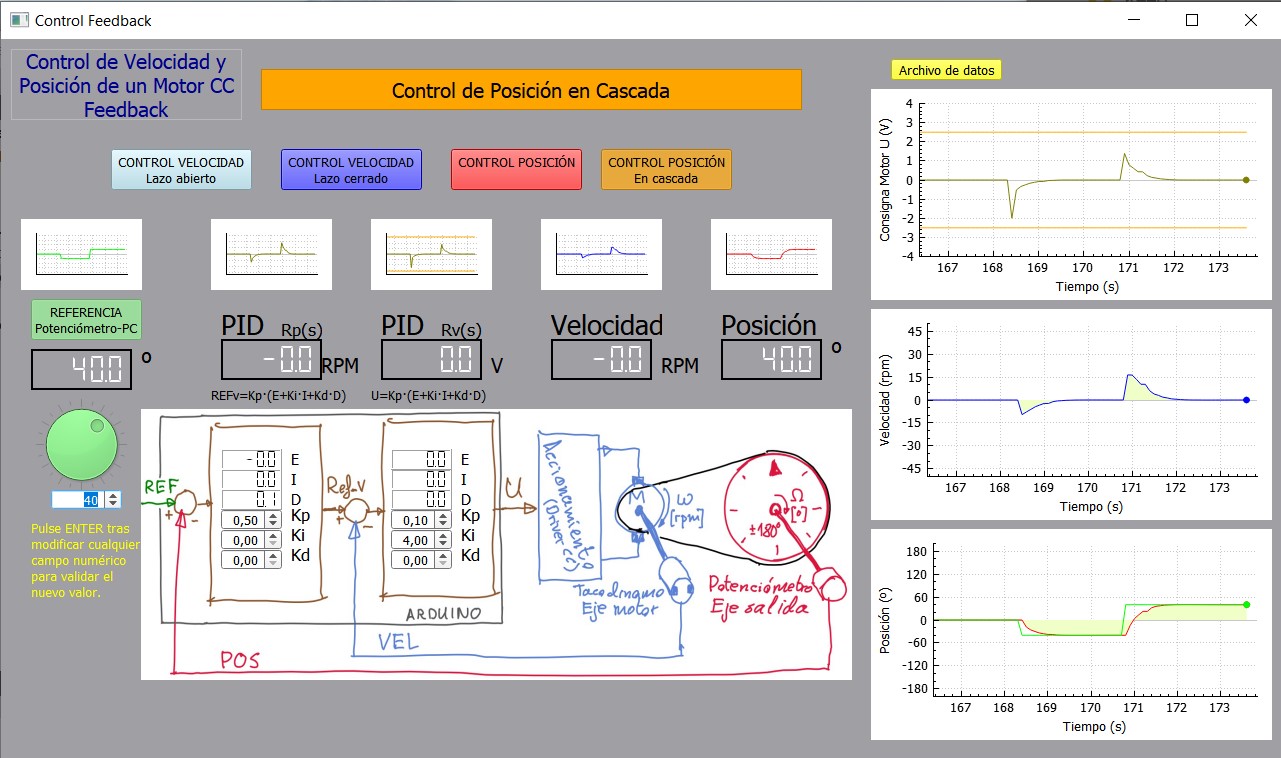
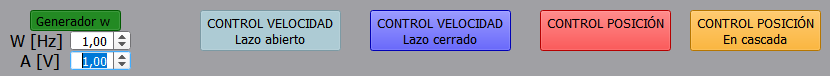
Para instalar el software en el Arduino es necesario disponer del software de programación que se puede encontrar en <https://www.arduino.cc/en/Main/Software> para diferentes plataformas Windows, MacOS y Linux.

Mediante este software se carga en el Arduino el programa “controlArduino.ino”.

### Instalación del programa Control-Feedback en un PC

El programa “Control-Feedback” se ha desarrollado para Windows y se puede instalar copiando la carpeta con el programa portable que contiene el ejecutable “Control-Feedback.exe”.

Para poder utilizar el programa es necesario que en el PC estén instalados los drivers de comunicación USB con el Arduino que, en el caso de haber utilizado el software de programación del Arduino disponible en <https://www.arduino.cc/en/Main/Software>, se habrán instalado por defecto.Si no es así, seguir las instrucciones de instalación de drivers de Arduino en https://www.arduino.cc/en/Guide/DriverInstallation.

****

## Guía de usuario

Los parámetros y variables del equipo de prácticas se pueden visualizar y controlar desde la aplicación Control-Feedback para PC, que se comunica con el microcontrolador Arduino a través de un puerto USB. El equipo se pone en marcha con el interruptor de encendido de la parte posterior e iniciando la aplicación Control-Feedback en el PC.

Imagen que contiene medidor

Descripción generada automáticamente Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Encender el

equipo

Iniciar la

aplicación

en el PC

En el caso de que no se disponga del equipo de prácticas o si este no está conectado a un puerto USB del PC, el programa presentará una ventana con un mensaje de error y una vez cerrada iniciará el programa Control-Feeback en modo de simulación (ver apartado 1.3).

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

El equipo de prácticas permite cuatro modalidades de control seleccionables mediante los pulsadores de la parte superior de la pantalla.



La señal de referencia en cualquiera de esas modalidades puede ser generada desde la unidad mecánica, mediante el potenciómetro conectado al eje de entrada, o desde la pantalla de la aplicación Control-Feedback. El conmutador verde “REFERENCIA” de la pantalla de la aplicación permite con una pulsación del ratón optar por un modo u otro de selección de la señal de referencia.

Imagen que contiene grande, reloj

Descripción generada automáticamenteUn reloj digital en la pantalla

Descripción generada automáticamente con confianza media

Conmutador de Referencia

generada desde el PC ->

o generada por el

potenciómetro de la

<- Unidad FeedBack

Valor de la Referencia ->

Selector de

<- Referencia

Introducción

<- numérica de la

Referencia

También es posible generar señales de referencia senoidales mediante el pulsador “Generador w”, situado sobre los controles anteriores, con dos campos numéricos que permiten fijar la frecuencia W (entre 0.1 y 2 Hercios) y la amplitud A (entre 0.1 y 2.5 Voltios). El generador anula la entrada de referencias mediante los controles del PC y de la unidad FeedBack hasta que se vuelva a pulsar en “Generador w”:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

En cualquiera de las modalidades de control se puede visualizar el valor numérico de las distintas variables y su evolución en el tiempo en las correspondientes gráficas.

Según el tipo de control que esté activo, se podrán modificar los parámetros de los diferentes reguladores PID que se utilicen y observar el valor calculado por estos mediante el algoritmo de control representado, que se aplica a los valores que tengan el Error a la entrada del regulador, la Integral del error y la Derivada del error.

Imagen de la pantalla de un celular de un mensaje en letras blancas

Descripción generada automáticamente con confianza baja

Valor de la consigna U para el

Motor, calculada por el PID ->

Error ->

Integral del error ->

Derivada del error ->

<- Algoritmo de cálculo de la

consigna U del PID

Parámetros PID modificables:

<- Valor de Kp

<- Valor de Ki

<- Valor de Kd

NOTA: Tras modificar cualquier campo numérico debe pulsarse ENTER para validar su nuevo valor.

En la parte superior derecha, sobre las gráficas, se encuentra un pulsador amarillo denominado: **Archivo de datos**. Al pulsar este pulsador se activa el registro de los datos procedentes del Arduino en un archivo ubicado por defecto en la carpeta “Documentos” de la carpeta del usuario: “Documentos/arduinodata\_yymmddhhmmss.csv”. Esta carpeta debe existir. Algunos sistemas operativos Windows antiguos no la reconocen por defecto y es necesario crearla para que se pueda generar el archivo “arduinodata.csv”.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| REF | R\_vel | PWM | VEL | Enc. | R\_pos | POS | Gray | E\_v | I\_v | D\_v | E\_p | I\_p | D\_p | BIN | milisegundos |
| 2.55 | 26.75 | 1.56 | 17.51 | 23.21 | 26.75 | 64.93 | 59.06 | 1.06 | 0.00 | 0.00 | 4.54 | 3.06 | 2.40 | 505 | 1554281123550 |

1970-01-01T00:00:00.000

2.55,26.75,1.56,17.51,23.21,26.75,64.93,59.06,1.06,0.00,0.00,4.54,3.06,2.40,505,1554281123550

2.55,26.75,1.56,17.50,23.25,26.75,68.63,67.82,1.06,0.00,0.00,4.54,3.06,2.40,504,1554281123613

2.55,26.75,1.56,17.55,23.24,26.75,72.12,70.16,1.06,0.00,0.00,4.54,3.06,2.40,240,1554281123691

…

Los datos aparecen en filas separados por comas incluyendo:

* REF: valor de la señal de referencia generada desde el potenciómetro del eje de entrada en voltios (0…5V).
* R\_vel: valor de la señal de referencia para el control de velocidad en rpm.
* PWM: señal de control enviada al accionamiento del motor (Driver CC) en valores de tensión (-2.5V: motor a máxima velocidad CCW, +2.5V: motor a máxima velocidad CW)). En realidad, la señal enviada al accionamiento del motor es una señal acotada a 0…5V resultado del filtrado de la señal que se describe para el conector   
  PWM de la placa de conexiones.
* VEL: velocidad en rpm del eje de salida. Positiva en el sentido de las agujas del reloj (CW) y negativa en el sentido contrario (CCW).
* Enc.: velocidad en rpm del eje de salida medida mediante el encoder incremental.
* R\_pos: valor de la señal de referencia para el control de posición en grados.
* POS: posición del eje de salida en grados tomando como posición 0º la vertical y variando desde -180º a 180º tomando como sentido positivo el giro en el sentido de las agujas del reloj (CW).
* Gray: posición en grados del eje de salida medida mediante el código de Gray.
* E\_v a D\_p: Error, Integral del error y Derivada del error de los reguladores de velocidad y posición respectivamente utilizados en cada instante para calcular las acciones de control.
* BIN: valor entero que representa el estado de los 9 bits correspondientes a los sensores digitales del equipo. Están ordenados desde la señal del diodo D1 (D1 del encoder incremental) como bit menos significativo, hasta el D9 (índice de paso por vuelta del disco del código de Gray) como bit más significativo.
* milisegundos: fecha y hora del registro de los datos de la línea medido en milisegundos transcurridos desde el 1 de enero de 1970 a las 00:00 horas.

## Modo de simulación

En el caso de que no se disponga del equipo de prácticas o si este no está conectado a un puerto USB del PC, el programa presentará una ventana con un mensaje de error y una vez cerrada iniciará el programa Control-Feeback en modo de simulación.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Se simulará el comportamiento del equipo utilizando como modelo del motor por defecto G(s)=K/(T·s+1)=16/(0.16·s+1) pudiendo modificarse los parámetros K y T mediante los campos numéricos que aparecen bajo el dibujo del motor:

Texto

Descripción generada automáticamente con confianza baja

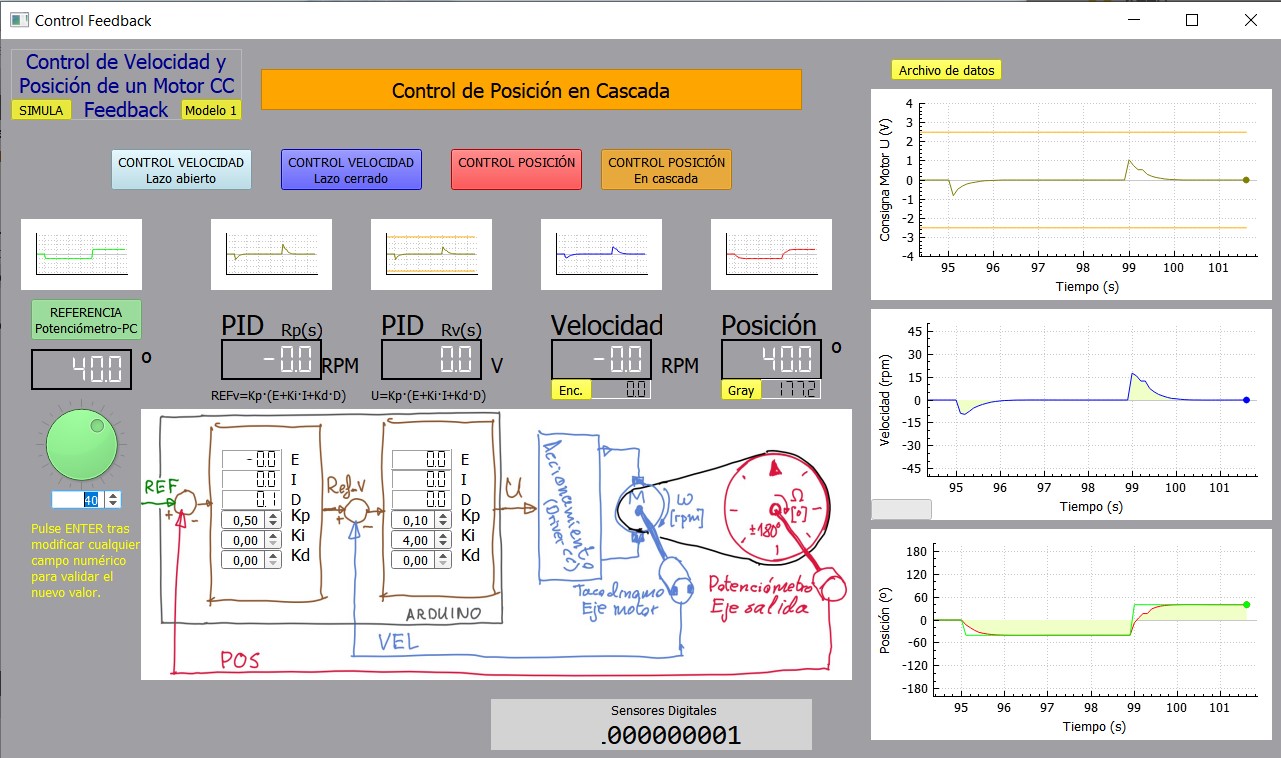
Todos los controles de la ventana funcionan de manera similar a como lo hacen con el equipo real conectado, salvo el selector “REFERENCIA Potenciómetro/PC” que no permite seleccionar la opción del potenciómetro de referencia del equipo. Además, aparece un control con un deslizador bajo el selector de referencia que permite simular el funcionamiento del freno magnético y una etiqueta en rojo que advierte del funcionamiento en modo de simulación:

Imagen que contiene reloj, medidor

Descripción generada automáticamente

## Controles avanzados

En la ventana de la aplicación se dispone de un pulsador “oculto” que permite acceder a algunos controles “avanzados” y al estado de los bits de los sensores digitales. Pulsando en la esquina inferior izquierda de la gráfica de la velocidad del motor aparecen estos controles en color amarillo. Volviendo a pulsar, desaparecen los controles, pero no las modificaciones que se hayan realizado con ellos.



Pulsador “oculto”

Índice de vuelta-^ ^-Gray--^ ^-^-encoder

Estado de los bits D1-D9 ---->987654321

- **NORMAL/SIMULA**: Este pulsador permite poner el Arduino en modo simulación de manera que el en lugar de leer la velocidad y posición del eje de los sensores, calcula estos valores basándose en modelos matemáticos de la maqueta. Es útil para hacer pruebas con un Arduino que no está conectado al equipo de Feedback.

- **MODELO 1/MODELO 0**: Su estado normal es con el texto “MODELO 1”. Si se pulsa y aparece el texto “MODELO 0”, el Arduino se configura para controlar equipos de Feedback que tienen la electrónica de control original del fabricante. Es necesario añadir un circuito de adaptación de señales entre el equipo de Feedback original y el Arduino para adaptar los niveles de tensión.

- **Enc.**: Este pulsador aparece acompañado por un display en el que se visualiza la velocidad en rpm del eje de salida medida por el encoder incremental del eje del motor. Si se pulsa, el sistema de control utiliza la velocidad medida por el encoder como realimentación en lugar de la señal de la tacodinamo.

- **Gray**: Este pulsador aparece acompañado por un display en el que se visualiza la posición en grados del eje de salida medida por el código de Gray del encoder absoluto del eje de salida. Si se pulsa, el sistema de control utiliza la posición medida por el código de Gray como realimentación en lugar de la señal del potenciómetro.

# Programa instalado en el Arduino: controlArduino.ino

//////////////////////////////////////////////////////////////////

// ########################################################## //

// ################### controlArduino ####################### //

// ########################################################## //

// //

// Versión 4.2 2/04/2022 12:00 //

// //

// Programa para el control de Velocidad y Posición de los //

// módulos de Feedback a través del interfaz gráfico en QT //

// //

//////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////

// Versión para los Arduino Micro. Para los Arduino MEGA //

// de Mecatrónica con la placa de adaptación de la Taco //

// con R2=56K R3=R4=100K, se deben modificar los valores de //

// las variables RPM\_max, V\_max y V\_0 en conversionAD() y //

// descomentar la línea que introduce un retardo de un //

// milisegundo en leerSerie(). //

// //

// Para el Arduino MEGA, hay varias partes del código //

// que no se utilizan porque no están cableadas las señales //

// las señales digitales del encoder ni del //

// del disco Gray. Por lo tanto, las funciones para las //

// interrupciones de los Canales A y B y la que calcula la //

// posición mediante el código Gray no se utilizan. //

//////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////

// //

// DENOMINACIÓN DE PINES //

// //

//////////////////////////////////////////////////////////////////

#define PIN\_velocidad A0 //Pin para la señal de velocidad del tacómetro

#define PIN\_posicion A1 //Pin para el potenciómetro de posición

#define PIN\_referencia A2 //Pin para el potenciómetro de referencia

// #define PIN\_PWM 3 //Pin para la salida de la señal PWM. La frecuencia de PWM 3 y 11 controlada con Timer0

#define PIN\_PWM 13 //Pin para la salida de la señal PWM para poder usar el 2 y el 3 para las interrupciones del encoder

//#define PIN\_D1 0 //Pin para el bit D1 del encoder Pin 0 interrupción RX. Entra en conflicto con el USB para interrupciones

//#define PIN\_D2 1 //Pin para el bit D2 del encoder Pin 1 interrupción TX. Entra en conflicto con el USB para interrupciones

#define PIN\_D1 2 //Pin para el bit D1 del encoder. Usar estos porque el Pin 0 y el 1 del puerto serie tienen conflicto con el USB

#define PIN\_D2 3 //Pin para el bit D2 del encoder.

// #define PIN\_D2 7 //Pin para el bit D2 del encoder. Para el Arduino con Pin 3 averiado

#define PIN\_D3 4 //Pin para el bit D3 del disco de Gray

#define PIN\_D4 5 //Pin para el bit D4 del disco de Gray

#define PIN\_D5 6 //Pin para el bit D5 del disco de Gray

#define PIN\_D6 7 //Pin para el bit D6 del disco de Gray

// #define PIN\_D6 12 //Pin para el bit D6 del disco de Gray. Para el Arduino con Pin 3 averiado

#define PIN\_D7 8 //Pin para el bit D7 del disco de Gray

#define PIN\_D8 9 //Pin para el bit D8 del disco de Gray

#define PIN\_D9 10 //Pin para el bit D9 del disco de Gray (índice de vuelta)

//////////////////////////////////////////////////////////////////

// //

// DECLARACIÓN DE VARIABLES //

// //

//////////////////////////////////////////////////////////////////

//Variables para temporización

long int tiempo\_old; // Variable que guarda el último valor de los milisegundos del reloj del Arduino.

long int tiempo\_new; // Variable que guarda el nuevo valor de los milisegundos del reloj del Arduino.

long int tiempo; // Variable en la que se calculan los milisegundos que han pasado.

long int tiempo2\_old; // Variable que guarda el último valor de los milisegundos del reloj del Arduino.

long int tiempo2\_new; // Variable que guarda el nuevo valor de los milisegundos del reloj del Arduino.

long int tiempo2; // Variable en la que se calculan los milisegundos que han pasado.

// Variables para medida digital de velocidad y posición

// \*\*\* Las variables usadas en interrupciones se recomienda declararlas como volátiles \*\*\*

volatile int contador; // Contador de pulsos + -> CW, - -> CCW

volatile bool A, B, Ao, Bo; // Estado de los bits del encoder

float vel\_enc=0; // Velocidad medida por el encoder

float pos\_gray=0; // Posición medida por el disco de Gray

//VARIABLES GLOBALES

int posicion; // Variable para el valor del potenciómetro de posición (int, 0...1023)

int referencia; // Variable para el valor del potenciómetro de referencia (int, 0...1023)

int velocidad; // Variable para el valor del tacómetro (int, 0...1023)

unsigned int binario; // Variable para transmitir los bits de los sensores digitales

unsigned int gray; // Variable para el código binario de Gray

bool bit1; // Bit D1 del disco del Encoder

bool bit2; // Bit D2 del disco del Encoder

bool bit3; // Bit D3 del disco de Gray

bool bit4; // Bit D4 del disco de Gray

bool bit5; // Bit D5 del disco de Gray

bool bit6; // Bit D6 del disco de Gray

bool bit7; // Bit D7 del disco de Gray

bool bit8; // Bit D8 del disco de Gray

bool bit9; // Bit D9 del disco de Gray (índice de vuelta)

float ref=0; // Valor de la referencia en voltios

float pos=0; // Valor de la posición en grados +-180º

float pos1=0;

float pos2=0;

float ref\_pos=0; //Referencia posición en grados +-180º

float vel=0; // Valor de la velocidad en rpm del eje de salida (eje de baja)

float vel1=0;

float ref\_vel=0; //Referencia velocidad en rpm del eje de salida (eje de baja)

int duty=127; // El Duty va de 0 (0%) a 255 (100%)

float pwm; // Valor del Duty en un float

float duty\_0 = 127.5; //Duty 50%, es decir, motor parado

float err\_v[3]; //Error entre la velocidad actual y la referencia

float err\_p[3]; //Error entre la posición actual y la referencia

float uk[3]; //Variables de salida del regulador PID

float W=1; //Frecuencia para el generador de senoides en Hercios

float Amp=1; //Amplitud de la senoide en voltios

// VARIABLES DE LAS CARACTERÍSTICAS DE LA TACODINAMO

float RPM\_max; // Máximas RPM CW del motor

float V\_max; // Tensión de la tacodinamo a las máximas RPM

float V\_0; // Tensión de la tacodinamo a 0 RPM

//VARIABLES PARA EL REGULADOR

float Tm=20; //Tiempo de muestreo en milisegundos

float T; //Tiempo real en ms desde la última ejecución de algoritmo de control()

//PID Velocidad

float kp\_v = 0.1; //Kp del PID

float ki\_v = 4.0; //Ki del PID

float kd\_v = 0.0; //Kd del PID

float integral\_v=0; //Integral del error

float derivada\_v=0; //Derivada del error

//PID Posición

float kp\_p = 0.5; //Kp del PID

float ki\_p = 0.0; //Ki del PID

float kd\_p = 0.0; //Kd del PID

float integral\_p=0; //Integral del error

float derivada\_p=0; //Derivada del error

//VARIABLES PARA RECIBIR DATOS SERIE

const byte buffSize = 32; //Tamaño de la cadena de caracteres

char input[buffSize]; //Cadena de caracteres recibida

byte maxChars = 31; //31 caracteres de máximo

int inputInt = 0; //Valor Int recibido

float inputFloat = 0.0; //Valor Float recibido

char inputCommand[12]; //Texto recibido

//VARIABLES PARA ELEGIR MODO DE CONTROL, REFERENCIA, MODELO, etc.

int modocontrol=4; // 1 vel lazo abierto, 2 vel lazo cerrado, 3 pos, 4 pos cascada

bool Pot\_PC=0; //Referencai del Potenciómetro=0 o del PC=1

bool modelo=1; //Modelo de unidad Feedback: 0->electrónica antigua, 1->electrónica modificada

bool sim=0; //Modo NORMAL=0, Modo SIMULACION=1

bool fb\_e=0; //Realimentación de velocidad por taco=0 o por encoder=1

bool fb\_g=0; //Realimentación de posición por potenciómetro=0 o por disco Gray=1

bool senoide=0; //Generador de funciones 0->desactivado, 1->activado

//////////////////////////////////////////////////////////////////

// //

// PROCEDIMIENTO leerSerie() //

// //

//////////////////////////////////////////////////////////////////

/\* Esta función lee los caracteres recibidos por el puerto serie (USB)

Espera tres cadenas de caracteres separados por comas,

los separa y los convierte al formato correcto de cada valor:

una cadena de caracteres, un valor entero y un valor real.

Todos serán guardados en diferentes variables. \*/

void leerSerie() {

input[0] = 0;

maxChars = buffSize - 1;

byte charCont = 0;

byte i = 0;

//Recibe los datos carácter a carácter y lo almacena en "input"

if (Serial.available() > 0) {

// delay(1); // Para dar tiempo a que llegue el comando completo 1ms

// (descomentar para el Arduino MEGA, en el MICRO no hace falta)

while (Serial.available() > 0) {

if (i > maxChars - 1) {

i = maxChars;

}

input[i] = Serial.read();

i ++;

charCont ++;

}

if (i > maxChars) {

i = maxChars;

}

input[i] = 0;

// DEVOLVER LO QUE ACABA DE RECIBIR PARA DEPURACION

// Serial.print(input);

// Serial.print("\n");

// Serial.flush();

//Ahora se debe dividir la cadena recibida en partes

char \* partirString; //Este puntero indicará donde cortar la cadena

partirString = strtok(input,","); //Se separa la primera parte - el texto

strcpy(inputCommand, partirString); //Lo copia en inputCommand

partirString = strtok(NULL, ","); //Se separa la segunda parte - el Int

inputInt = atoi(partirString); //Lo convierte a int

partirString = strtok(NULL, ","); //Se separa la segunda parte - el Float

inputFloat = atof(partirString); //Lo convierte a float

//////////////////////////////////////////////////////////////////

// Interpretación y ejecución del comando recibido //

//////////////////////////////////////////////////////////////////

/\* Interpretación de los datos recibidos y operación según el comando recibido desde el PC.

Comandos aceptados: Todos los comandos pueden ir seguidos de un "int" y un "float" separados

por comas:

cmd,entero,real,

Comandos sin parámetros:

dat - Petición de datos desde el PC. Se le enviarán los valores de las variables.

rst - Se pone el programa con los parámetros por defecto. Petición desde el PC al iniciar o cerrar la conexión.

est - Petición desde el PC del estado de los modos de control, referencia, realimentacion, etc.

Comandos con parámetro entero binario (0<>1):

mod - Selecciona el modelo de Feedback original=0, modificado=1

sim - Funcionamiento en modo normal=0, simulación=1

ppc - Selecciona la señal de referencia desde el potenciómetro de la unidad Feedback=0 o desde el PC=1

fbv - Realimentación de velocidad por tacodinamo=0 o encoder=1

fbp - Realimentación de posición por potenciometro=0 o disco Gray=1

Comandos con parámetro entero:

con - Modo de control: vel lazo abierto=1, vel lazo cerrado=2, pos un lazo simple=3, pos en cascada=4

Comandos con parámetros entero y real:

ref - Valor de la referencia en el valor "float" desde el PC. El "int" indica referencia de velocidad=0 o posición=1

pid - Valores de las constantes de los reguladores PID en el valor "float". El "int" indica la constante a modificar:

1=Kp del regulador de velocidad

2=Ki del regulador de velocidad

3=Kd del regulador de velocidad

4=Kp del regulador de posición

5=Ki del regulador de posición

6=Kd del regulador de posición

sin - Generador de referencias senoidales. Valor del "int":

0=desactivar

1=activar

2=valor de W en hercios (el valor "float" indica la W frecuencia en hercios)

3=valor de A en voltios (el valor "float" indica la A amplitud en voltios)

\*/

//PETICION DE DATOS

if (strcmp (inputCommand,"dat") == 0){

envioDatos();

}

//PETICION DE ESTADO PARA DEPURACION

if (strcmp (inputCommand,"est") == 0){

envioestado();

}

//INICIO O CIERRE DE COMUNICACIÓN

//Paso a control de Posición con el Potenciómetro

if (strcmp (inputCommand,"rst") == 0){

Pot\_PC=0;

modocontrol=4; // Control de posición con lazo en cascada

sim=0; //Quitar modo SIMULACION si está puesto

senoide=0; // Desactiva el generador de senoides

fb\_e=0;

fb\_g=0;

kp\_v=0.1; //Valores por defecto para los reguladores

ki\_v=4;

kd\_v=0.0;

kp\_p=0.5;

ki\_p=0.0;

kd\_p=0.0;

integral\_v=0;

integral\_p=0;

}

//MODELO DE UNIDAD FEEDBACK ANTIGUO (0) O NUEVO (1)

if (strcmp (inputCommand,"mod") == 0){

modelo=inputInt;

}

//FUNCIONAMIENTO NORMAL (0) O EN SIMULACIÓN (1)

if (strcmp (inputCommand,"sim") == 0){

sim=inputInt;

}

//REFERENCIA POTENCIOMETRO (0) O PC (1)

if (strcmp (inputCommand,"ppc") == 0){

Pot\_PC=inputInt;

ref\_pos=0;

ref\_vel=0;

integral\_v=0;

integral\_p=0;

}

//REALIMENTACIÓN DE VELOCIDAD POR TACODINAMO (0) O ENCODER (1)

if (strcmp (inputCommand,"fbv") == 0){

fb\_e=inputInt;

}

//REALIMENTACIÓN DE POSICIÓN POR POTENCIÓMETRO (0) O GRAY (1)

if (strcmp (inputCommand,"fbp") == 0){

fb\_g=inputInt;

}

//MODO DE CONTROL

if (strcmp (inputCommand,"con") == 0){

if ((Pot\_PC==1) & (modocontrol>2) & (inputInt<3)) ref\_vel=0.25\*ref\_pos; // Cambio de referencia de posición a referencia de velocidad enviada por el PC

if ((Pot\_PC==1) & (modocontrol<3) & (inputInt>2)) ref\_pos=4\*ref\_vel; // Cambio de referencia de velocidad a referencia de posición enviada por el PC

modocontrol=inputInt; // vel lazo abierto=1, vel lazo cerrado=2, pos un lazo simple=3, pos en cascada=4

integral\_v=0;

integral\_p=0;

}

//NUEVA REFERENCIA

if (strcmp (inputCommand,"ref") == 0){

if (inputInt) ref\_pos=inputFloat; // El entero indica si cambia la referencia de velocidad (0) o de posición (1)

else ref\_vel=inputFloat; // Valor de la nueva referencia

}

//CAMBIO VALORES DE LOS REGULADORES PID

if (strcmp (inputCommand,"pid") == 0){

switch (inputInt) { // El entero indica que constante se va a cambiar

case 1:

kp\_v=inputFloat; // Regulador PID de velocidad

break;

case 2:

ki\_v=inputFloat;

break;

case 3:

kd\_v=inputFloat;

break;

case 4:

kp\_p=inputFloat; // Regulador PID de posición

break;

case 5:

ki\_p=inputFloat;

break;

case 6:

kd\_p=inputFloat;

break;

default:

// if nothing else matches, do the default

// default is optional

break;

}

}

//GENERADOR DE SENOIDES

if (strcmp (inputCommand,"sin") == 0){

switch (inputInt) { // El entero indica que valor se va a cambiar

case 0:

senoide=0; // Regulador PID de velocidad

break;

case 1:

senoide=1; // Regulador PID de velocidad

break;

case 2:

W=inputFloat;

break;

case 3:

Amp=inputFloat;

break;

default:

// if nothing else matches, do the default

// default is optional

break;

}

}

}

}

//////////////////////////////////////////////////////////////////

// //

// PROCEDIMIENTO envioDatos() //

// //

//////////////////////////////////////////////////////////////////

/\* Con esta función se envían los datos por el puerto serie USB hacia el PC.

Los datos van en una cadena separados por comas:

"referencia,ref\_vel,pwm,velocidad,velocidad\_enc,ref\_pos,posición,posición\_gray,error\_v,int\_v,der\_v,error\_p,int\_p,der\_p,binario"

referencia - valor de referencia del Potenciómetro en voltios

ref\_vel - valor de referencia del velocidad en rpm

pwm - salida del regulador hacia el accionamiento del motor

velocidad - velocidad del eje de baja en rpm

velocidad\_enc - velocidad calculada con el código digital del encoder

posición - posición en grados del eje de baja (-180º a +180º)

posición\_gray - posición calculada con el código digital de Gray

error\_v - error a la entrada del regulador de velocidad

int\_v - integral del error de velocidad

der\_v- derivada del error de velocidad

error\_p - error a la entrada del regulador de posición

int\_p - integral del error de posición

der\_p- derivada del error de posición

binario- valor binario de las entradas digitales (Encoder,Gray,Bit de paso por vuelta)

La función se ejecuta cada vez que se piden los datos desde el PC con el comando "dat"

\*/

void envioDatos(){

//Enviar el dato de REFERENCIA DEL POTENCIÓMETRO en voltios

Serial.print(ref);

Serial.print(",");

//Enviar el dato de REFERENCIA DE VELOCIDAD

Serial.print(ref\_vel);

Serial.print(",");

//Enviar el dato de Control del Motor (salida del PID)

Serial.print(uk[0]);

Serial.print(",");

//Enviar el dato de VELOCIDAD

Serial.print(vel);

Serial.print(",");

//Enviar datos digitales de velocidad encoder

Serial.print(vel\_enc);

Serial.print(",");

//Enviar el dato de REFERENCIA de POSICION

Serial.print(ref\_pos);

Serial.print(",");

//Enviar el dato de POSICIÓN

Serial.print(pos);

Serial.print(",");

//Enviar datos digitales de Posición Gray

Serial.print(pos\_gray);

Serial.print(",");

//Enviar el dato de error de velocidad

Serial.print(err\_v[0]);

Serial.print(",");

//Enviar el dato de integral

Serial.print(integral\_v);

Serial.print(",");

//Enviar el dato de derivada

Serial.print(derivada\_v);

Serial.print(",");

//Enviar el dato de error de posición

Serial.print(err\_p[0]);

Serial.print(",");

//Enviar el dato de integral

Serial.print(integral\_p);

Serial.print(",");

//Enviar el dato de derivada

Serial.print(derivada\_p);

Serial.print(",");

//Enviar el dato de entradas digitales

Serial.print(binario);

Serial.print(",");

// Enviar todo

Serial.flush();

}

//////////////////////////////////////////////////////////////////

// //

// PROCEDIMIENTO envioestado() //

// //

//////////////////////////////////////////////////////////////////

/\* Envío del estado de las variables que controlan el modo de funcionamiento

(Para depuración)

\*/

void envioestado(){

//Enviar el dato de estado

Serial.print("Modo\_Control=");

Serial.print(modocontrol); // Control: vel lazo abierto=1, vel lazo cerrado=2, pos un lazo simple=3, pos en cascada=4

Serial.print(",Potenciometro\_PC=");

Serial.print(Pot\_PC); // Referencia desde el potenciómetro de la unidad Feedback=0 o desde el PC=1

Serial.print(",modelo=");

Serial.print(modelo); // Modelo de Feedback original=0, modificado=1

Serial.print(",simulacion=");

Serial.print(sim); // Funcionamiento en modo normal=0, simulación=1

Serial.print(",encoder=");

Serial.print(fb\_e); // Realimentación de velocidad por tacodinamo=0 o encoder=1

Serial.print(",gray=");

Serial.print(fb\_g); // Realimentación de posición por potenciometro=0 o disco Gray=1

Serial.print(",");

// Enviar todo

Serial.flush();

}

//////////////////////////////////////////////////////////////////

// //

// FUNCIÓN gray2binary() //

// //

//////////////////////////////////////////////////////////////////

/\*

Función para convertir un entero código de GRAY en el entero binario correspondiente:

gray - entero del código de GRAY

\*/

unsigned int gray2binary(unsigned int gray)

{

unsigned int num, mask;

num = gray;

mask = num >> 1; // Desplaza un bit a la derecha

while (mask > 0) {

num = num^mask; // XOR entre num y mask

mask = mask >> 1; // Desplaza un bit a la derecha

}

return num;

}

//////////////////////////////////////////////////////////////////

// //

// PROCEDIMIENTOS CanalA() y CanalB() //

// //

//////////////////////////////////////////////////////////////////

// Rutinas de interrupción para el encoder digital

void CanalA(){ // Flanco en el canal A

// A=!A; // estas dos líneas no funcionan bien ¿por qué?

// contador=contador-A\*B; // sería un código más óptimo que el actual

A=digitalRead(PIN\_D2); // Actualizar el estado correcto del pin del encoder

if (A==!Ao) contador=contador-A\*B; // pulso en sentido CCW si A=B=1 cuando cambia A

Ao=A;

}

void CanalB(){ // Flanco en el canal B

// B=!B;

// contador=contador-A\*B;

B=digitalRead(PIN\_D1);

if (B==!Bo) contador=contador+A\*B; // pulso en sentido CW si A=B=1 cuando cambia B

Bo=B;

}

//////////////////////////////////////////////////////////////////

// //

// PROCEDIMIENTO conversionAD() //

// //

//////////////////////////////////////////////////////////////////

/\* Función mediante la cual se leerán las entradas

analógicas y digitales con las que se trabaja.

\*/

void conversionAD(){

/\* La velocidad y posición del equipo se estiman a partir de la medida de los sensores, pero si

se trabaja en modo simulación ambas se calculan simulando la función de transferencia del sistema

\*/

if (sim) { // Modo SIMULACION el Arduino simula los valores de velocidad y posición.

if (uk[0]>2.5) uk[0]=2.5; // Limitado para simulación

if (uk[0]<-2.5) uk[0]=-2.5;

if (uk[1]>2.5) uk[1]=2.5; // Limitado para simulación

if (uk[1]<-2.5) uk[1]=-2.5;

if (uk[2]>2.5) uk[2]=2.5; // Limitado para simulación

if (uk[2]<-2.5) uk[2]=-2.5;

vel=0.8521\*vel1+(255/5)\*0.04639\*uk[1]; // Ecuación en diferencias de la G(z) del accionador/motor/reductora para velocidad

if (vel>45) vel=45; // Discretizada para Tm=20ms

if (vel<-45) vel=-45;

pos=pos+6\*vel1\*T/1000;

// pos=1.8521\*pos1-0.8521\*pos2+(255/5)\*(0.02857\*uk[1]+0.02709\*uk[2]);

if (pos>180) pos=pos-360; // Ecuación en diferencias de la G(z) del accionador/motor/reductora para posición

if (pos<-180) pos=pos+360; // Discretizada para Tm=20ms

vel1=vel;

pos1=pos;

pos2=pos1;

}

else { // Si no está en modo simulación, valores medidos de los sensores.

/\* Entradas analógicas 0...5V a un entero 0...1023 (10bits) \*/

referencia= analogRead(PIN\_referencia); // Valor entero sin signo 0...1023

posicion= analogRead(PIN\_posicion); // Valor entero sin signo 0...1023

velocidad= analogRead(PIN\_velocidad); // Valor entero sin signo 0...1023

/\* Se tratarán la señales analógicas y se transforman de valores enteros (int)

a valores reales (float) relacionados con la magnitud que se mide. \*/

//POSICION

pos=(float) posicion\*(360.0/1023.0)-180.0; // "posicion" en int 0...1023 que hay que pasar a "pos" en grados +-180º

if (fb\_g) pos=pos\_gray; // Realimentación de posición por código de GRAY

//VELOCIDAD

/\* El valor estimado de la velocidad depende de las carácteristicas de cada tacodinamo

y de los valores de las resistencias de la placa que adapta su tensión de salida (+-10V teóricos)

a la tensión que admite el ARDUINO (0...5V).

La mejor forma de que la conversión se aproxime a la velocidad real es medir la tensión del borne

"velocidad" de la placa cuando el motor está parado "V\_0" y cuando este gira a velocidad máxima CW

"V\_max", anotando cual es esa velocidad máxima en RPM "RPM\_max\*/

// Valores aproximados para R2=47K R3=R4=100K en placas con Arduino MICRO

RPM\_max=45; // Máximas RPM CW del motor

V\_max=3.5; // Tensión del motor a las máximas RPM

V\_0=2.55; // Tensión de la tacodinamo a 0 RPM

// Valores aproximados para R2=56K R3=R4=100K en placas con Arduino MEGA

// RPM\_max=45; // Máximas RPM CW del motor

// V\_max=3.35; // Tensión del motor a las máximas RPM

// V\_0=2.36; // Tensión de la tacodinamo a 0 RPM

// Valores aproximados si en las placas R2=50K R3=R4=100K

// RPM\_max=45; // Máximas RPM CW del motor

// V\_max=3.44; // Tensión del motor a las máximas RPM

// V\_0=2.5; // Tensión de la tacodinamo a 0 RPM

vel=(float) velocidad\*(5.0/1023.0); // "vel" en voltios (de 0...1023 a 0V...5V) que hay que pasar a RPM

vel=(vel-V\_0)\*(RPM\_max/(V\_max-V\_0)); // paso de voltios a RPM

if (fb\_e) vel=vel\_enc; // Realimentación de velocidad por encoder

}

//REFERENCIA

ref=(float) referencia\*(5.0/1023.0); // referencia (0...1023) a ref en voltios (0V...5V).

//Se comprueba el modo de control (posición o velocidad) y si la referencia viene del potenciómetro o del PC

//Si la referencia la envía el PC, Pot\_PC==1, no se actualiza con la de la entrada A/D "referencia".

if (Pot\_PC==0 && modocontrol<3){ // modos de control de velocidad

ref\_vel=(float) referencia\*(90.0/1023.0)-45.0; // Paso 0...1023 a -45...+45 rpm

}

if (Pot\_PC==0 && modocontrol>2){ // modos de control de posición

ref\_pos=(float) referencia\*(360.0/1023.0)-180.0; // Paso 0...1023 a -180º...+180º

}

// ENTRADA SENOIDAL

if (senoide==1) { //Si se selecciona senoidal senoide=1

// float Amp=2; //Amplitud en voltios (entre +-2.5V)

// float W=1; //Frecuencia en hercios: 1[hercio]=1[ciclos/s]=2\*pi[rad/s]

ref=Amp\*sin(2\*3.1416\*W\*tiempo\_new/1000.0)+2.5; // tiempo\_new en milisegundos, ref 0...5V

referencia=ref\*1023.0/5.0;

ref\_vel=(float) referencia\*(90.0/1023.0)-45.0;

ref\_pos=(float) referencia\*(360.0/1023.0)-180.0;

}

/\* Entradas digitales del Encoder, Gray e Indice de vuelta \*/

bit1=digitalRead(PIN\_D1); // Encoder

bit2=digitalRead(PIN\_D2);

bit3=digitalRead(PIN\_D3); // Gray

bit4=digitalRead(PIN\_D4);

bit5=digitalRead(PIN\_D5);

bit6=digitalRead(PIN\_D6);

bit7=digitalRead(PIN\_D7);

bit8=digitalRead(PIN\_D8);

bit9=digitalRead(PIN\_D9); // Indice de vuelta

// Se combinan todas en una única variable con todos los bits para el envío al PC

binario=bit1+2\*bit2+4\*bit3+8\*bit4+16\*bit5+32\*bit6+64\*bit7+128\*bit8+256\*bit9;

gray=bit8+2\*bit7+4\*bit6+8\*bit5+16\*bit4+32\*bit3; // Código de GRAY

pos\_gray=-(gray2binary(gray)-31.5)\*360/64; // Posición en grados del código de GRAY

}

//////////////////////////////////////////////////////////////////

// //

// PROCEDIMIENTO conversionDA() //

// //

//////////////////////////////////////////////////////////////////

/\* Función mediante la cual se genera el Duty para la salida PWM

a partir de la salida del regulador. \*/

void conversionDA(){

// El duty va de 0 a 255 (0% a 100%, duty\_0=127.5 => motor parado=50%) mientras que

// la salida del regulador uk[0] varía de -2.5 a +2.5V (o más, no está limitada).

pwm=uk[0]\*255/5; // Paso de -2.5...+2.5V a -127.5...+127.5

if (pwm>127.5) pwm=127.5;

if (pwm<-127.5) pwm=-127.5;

// Además, como el motor tiene una zona muerta de duty, se trata de evitar sumando (duty de giro CW) o

// restando al duty (duty de giro CCW) una cantida aproximadamente equivalente a esa

// zona muerta (+-20 para el modelo 1 y +-15 para el modelo 0) pero manteniendo la linealidad (127.5-20)/127.5.

if (modelo) { // Si el MODELO es el nuevo el duty al 100% gira CW => duty = uk[0]+duty\_0;

if (pwm>0) duty=(pwm\*(127.5-20)/127.5)+20+duty\_0;

else if (pwm<0) duty=(pwm\*(127.5-20)/127.5)-20+duty\_0;

}

else { // Si el MODELO es el antiguo el duty al 100% gira CCW => duty = -uk[0]+duty\_0;

if (pwm>0) duty=-((pwm\*(127.5-15)/127.5)+15)+duty\_0;

else if (pwm<0) duty=-((pwm\*(127.5-15)/127.5)-15)+duty\_0;

}

Esquemático

Descripción generada automáticamente

// Limitador de duty (mejor limitar "pwm" para evitar errores de paso de real (float) a entero (int)

//if (duty<0) duty = 0; // 0%

//else if (duty>255) duty=255; // 100%

// En control de velocidad, parada del motor si la referencia de velocidad es inferior a +-2 r.p.m.

// Si no, el motor con ref\_vel=0 gira ligeramente por errores de medida del tacogenerador

if ((modocontrol<3) && (ref\_vel<2) && (ref\_vel>-2)) {

duty = duty\_0;

integral\_v=0;

}

//Envío del valor del duty al puerto PWM conectado al driver del motor.

analogWrite(PIN\_PWM,duty);

}

//////////////////////////////////////////////////////////////////

// //

// PROCEDIMIENTO regulador() //

// //

//////////////////////////////////////////////////////////////////

/\* En esta función se recogen los algoritmos de control.

El control puede ser:

+-LAZO ABIERTO (modocontrol=1)

+

+ +-UN LAZO (modocontrol=3)

+ +-POSICIÓN-+

+ + +-EN CASCADA (modocontrol=4)

+-LAZO CERRADO-+

+

+-VELOCIDAD ((modocontrol=2) o (modocontrol=4))

\*/

void regulador(){

//MODO LAZO ABIERTO

if (modocontrol==1) {

uk[0]=ref\_vel\*5/90; // La consigna del "regulador" será la referencia de velocidad +-45 RPM pasada a +-2.5V

}

//MODO LAZO CERRADO

// CONTROL DE POSICIÓN

if (modocontrol>2){

//CÁLCULO DEL ERROR

// Referencia del potenciómetro

if (Pot\_PC==0) {

err\_p[0] = ref\_pos - pos;

}

// Referencia del PC

else {

if (((ref\_pos-pos)<180.0)&&((ref\_pos-pos)>-180.0)) { // Problema del paso +180º -180º

err\_p[0] = ref\_pos - pos;

} // Busca el camino más corto para llegar a la nueva posición

else if ((ref\_pos-pos)>180.0) { // paso de -180º a +180º

err\_p[0] = ref\_pos - pos - 360;

}

else if ((ref\_pos-pos)<-180.0) { // paso de +180º a -180º

err\_p[0] = ref\_pos - pos + 360;

}

}

//CÁLCULO DE LA INTEGRAL DEL ERROR

//Sistema anti-windup

if (duty<1 || duty>254 ) // Saturación

{

integral\_p=integral\_p;

// integral\_p=0;

}

else // si no hay saturación se calcula el valor de la integral normalmente

{

integral\_p=integral\_p+err\_p[1]\*T/1000; // T está en ms, hay que dividirlo por 1000 para pasarlo a s

}

// Si no hay acción integral, se resetea el valor de la integral

if (ki\_p==0) integral\_p=0;

//CÁLCULO DE LA DERIVADA DEL ERROR

derivada\_p=1000\*(err\_p[0]-err\_p[1])/T; // T está en ms, hay que dividirlo por 1000 para pasarlo a s

//CÁLCULO DEL ALGORITMO DEL PID

if (modocontrol==3) { // Control de posición lazo simple: se calcula la consigna para el motor

uk[0]=kp\_p\*(err\_p[0]+ki\_p\*integral\_p+kd\_p\*derivada\_p);

}

else { // Control de posición en cascada: se calcula la referencia de velocidad

ref\_vel=kp\_p\*(err\_p[0]+ki\_p\*integral\_p+kd\_p\*derivada\_p);

}

}

// CONTROL DE VELOCIDAD (si se hace control de velocidad en lazo cerrado y si se hace control de posición en cascada)

if ((modocontrol==2) || (modocontrol==4)) {

//CÁLCULO DEL ERROR

err\_v[0] = ref\_vel - vel; // Si el control es de posición en cascada ref\_vel lo ha generado el regulador de posición

//CÁLCULO DE LA INTEGRAL DEL ERROR

//Sistema anti-windup

if (duty<1 || duty>254 ) // Saturación

{

integral\_v=integral\_v;

// integral\_v=0;

}

else // si no hay saturación se calcula el valor de la integral normalmente

{

integral\_v=integral\_v+err\_v[1]\*T/1000; // T está en ms, hay que dividirlo por 1000 para pasarlo a s

}

// Si no hay acción integral, se resetea el valor de la integral

if (ki\_v==0) integral\_v=0;

//CÁLCULO DE LA DERIVADA DEL ERROR

derivada\_v=1000\*(err\_v[0]-err\_v[1])/T; // T está en ms, hay que dividirlo por 1000 para pasarlo a s

//CÁLCULO DEL ALGORITMO DEL PID

uk[0]=kp\_v\*(err\_v[0]+ki\_v\*integral\_v+kd\_v\*derivada\_v);

}

// Desplazar valores

err\_v[2]= err\_v[1];

err\_v[1]= err\_v[0];

err\_p[2]= err\_p[1];

err\_p[1]= err\_p[0];

uk[1]= uk[0];

uk[2]= uk[1];

}

//////////////////////////////////////////////////////////////////

// ########################################################## //

// ##################### SETUP() ########################### //

// ########################################################## //

//////////////////////////////////////////////////////////////////

void setup() {

Serial.begin(115200); // Velocidad en bits por segundo del puerto serie USB. Resto de parámetros por defecto

//Iniciar a cero las tablas

err\_v[0]=0;

err\_v[1]=0;

err\_v[2]=0;

err\_p[0]=0;

err\_p[1]=0;

err\_p[2]=0;

uk[0]=0;

uk[1]=0;

uk[2]=0;

//Modo entrada/salida de cada PIN

pinMode(PIN\_posicion, INPUT);

pinMode(PIN\_referencia, INPUT);

pinMode(PIN\_velocidad, INPUT);

pinMode(PIN\_PWM, OUTPUT);

// pinMode(PIN\_D1, INPUT); // Si se declaran simplemente como INPUT no funcionan con el encoder !!!

// pinMode(PIN\_D2, INPUT); // Si se declaran simplemente como INPUT no funcionan con el encoder !!!

pinMode(PIN\_D1, INPUT\_PULLUP); // Para usarlos en interrupciones: INPUT\_PULLUP

pinMode(PIN\_D2, INPUT\_PULLUP); // Incluso si no se usan interrupciones es necesario!!!

// Si se usan interrupciones hay que declarar el PIN asociado con las dos siguientes líneas

attachInterrupt(digitalPinToInterrupt(PIN\_D1),CanalB,CHANGE); // Función que atiende la interrupción CanalB()

attachInterrupt(digitalPinToInterrupt(PIN\_D2),CanalA,CHANGE); // Función que atiende la interrupción CanalA()

pinMode(PIN\_D3, INPUT);

pinMode(PIN\_D4, INPUT);

pinMode(PIN\_D5, INPUT);

pinMode(PIN\_D6, INPUT);

pinMode(PIN\_D7, INPUT);

pinMode(PIN\_D8, INPUT);

pinMode(PIN\_D9, INPUT);

A=digitalRead(PIN\_D2);

B=digitalRead(PIN\_D1);

Ao=A;

Bo=B;

tiempo\_old = millis(); // Inicia el primer valor de los milisegundos del reloj del Arduino.

tiempo2\_old = millis(); // Inicia el primer valor de los milisegundos del reloj del Arduino.

// FRECUENCIA DEL PWM

// SOLO PARA Arduino MICRO con ATMega32U4. Ver para otros Arduinos y Procesadores

// AHORA SE HA CAMBIADO EL PWM AL PIN 13 PARA USAR EL 2 Y EL 3 PARA LAS INTERRUPCIONES DEL ENCODER

// TCCR0B = TCCR0B & 0b11111000 | 0x03;

//Frecuencia del PWM para pin 3 y 11: Timer0

// 0x01 62500 Hz 16MHz/(1\*256) (16MHz system Clock)

// 0x02 7812 Hz 16MHz/(8\*256)

// 0x03 976 Hz 16MHz/(64\*256) (por defecto)

// 0x04 244 Hz 16MHz/(256\*256)

// 0x05 61 Hz 16MHz/(1024\*256)

//

// Nota: -La electrónica de Feedback recomienda una frecuencia de 300Hz (244Hz es bastante ruidosa).

// Funciona mejor a 7812Hz o 976Hz a pesar de emitir un pitido molesto.

// -El controlador de motores MD03 solicita una frecuencia de 20KHz o más.

// Sin embargo funciona correctamente con el modo por defecto (976Hz) y no produce ruido.

// -El controlador de motores MD22 no especifica que soporte señal PWM.

// Funciona correctamente con el modo por defecto pero es conveniente poner un filtro RC para evitar el ruido.

//

}

//////////////////////////////////////////////////////////////////

// ########################################################## //

// ###################### LOOP() ########################### //

// ########################################################## //

//////////////////////////////////////////////////////////////////

void loop() {

//Bucle de ejecución continua

leerSerie(); // Comprobar si han llegado órdenes por el puerto serie

// Se controla el tiempo que ha pasado desde la anterior ejecución para calcular la velocidad del encoder cada 100ms

tiempo2\_new = millis(); // Miramos el nuevo valor de los milisegundos del reloj.

tiempo2 = tiempo2\_new - tiempo2\_old; // Calculamos los milisegundos que han pasado.

if (tiempo2 > 100) // "tiempo2" debería ser positivo salvo OVERFLOW de las variables que son "long int"

{ // (no se ha programado el tratamiento del OVERFLOW)

vel\_enc=234.375\*contador/tiempo2; //Cálculo de la velocidad por el contador del encoder cada 100ms

// N[pulsos/ms]\*[1000ms/1s]\*[60s/1min]\*[1vuelta/(8\*32)pulsos]=N\*1000\*60/(8\*32)[vuelta/min]=N\*234.375[rpm]

contador=0; // Se pone a cero el contador de pulsos incrementado por las interrupciones de los canales A y B

tiempo2\_old=tiempo2\_new;

}

// Se controla el tiempo que ha pasado desde la anterior ejecución del "control()" para ejecutar el

// algoritmo de control cada Tm ms aproximadamente y a ajustar el "T" de cálculo del algoritmo de control

tiempo\_new = millis(); // Se mira el nuevo valor de los milisegundos del reloj.

tiempo = tiempo\_new - tiempo\_old; // Se calculan los milisegundos que han pasado.

if (tiempo > Tm-1) // "tiempo" debería ser positivo salvo OVERFLOW de las variables que son "long int"

{ // (no se ha programado el tratamiento del OVERFLOW)

T=tiempo; //Se ajusta el tiempo del cálculo del PID al tiempo transcurrido en ms que, en principio, debería ser Tm ms

conversionAD(); // Lectura de valores de sensores y referencia

regulador(); // Ejecución del algoritmo de control que proceda

conversionDA(); // Envío de la señal de control a la salida PWM

tiempo\_old=tiempo\_new;

}

}

# Protocolo de comunicación

El programa controlArduino mantiene un protocolo de comunicación Cliente/Servidor a través de un puerto serie USB en modo Maestro/Esclavo. La comunicación la inicia el equipo Cliente, el PC, pidiendo al programa del Arduino mediante un comando la modificación de algún parámetro del programa o la respuesta con un mensaje. Los comandos se envían en forma de cadena de caracteres a través del puerto serie configurado a una velocidad de 115200 baudios. Todos los comandos pueden ir seguidos de dos parámetros, un "int" y un "float", separados por comas: “cmd,entero,real,”

Comandos sin parámetros:

dat - Petición de datos desde el PC. Se le enviarán los valores de las variables.

rst - Se pone el programa con los parámetros por defecto. Petición del PC al iniciar o cerrar la conexión.

est - Petición desde el PC del estado de los modos de control, referencia, realimentacion, etc.

Comandos con parámetro entero binario (0<>1):

mod - Selecciona el modelo de Feedback original=0, modificado=1

sim - Funcionamiento en modo normal=0, simulación=1

ppc - Selecciona la señal de referencia desde el potenciómetro de la unidad Feedback=0 o desde el PC=1

fbv - Realimentación de velocidad por tacodinamo=0 o encoder=1

fbp - Realimentación de posición por potenciometro=0 o disco Gray=1

Comandos con parámetro entero:

con - Modo de control: vel lazo abierto=1, vel lazo cerrado=2, pos un lazo simple=3, pos en cascada=4

Comandos con parámetros entero y real:

ref - Valor de la referencia en el valor "float" del PC. El "int" indica ref. de velocidad=0 o posición=1

pid - Valores de las constantes de los PID en el valor "float". El "int" indica la constante a modificar:

1=Kp del regulador de velocidad

2=Ki del regulador de velocidad

3=Kd del regulador de velocidad

4=Kp del regulador de posición

5=Ki del regulador de posición

6=Kd del regulador de posición

sin - Generador de referencias senoidales. Valor del "int":

0=desactivar

1=activar

2=valor de W en hercios (el valor "float" indica la W frecuencia en hercios)

3=valor de A en voltios (el valor "float" indica la A amplitud en voltios)

La respuesta hacia el PC al comando “dat” enviado por este es una cadena de caracteres con valores numéricos separados por comas:

"referencia,ref\_vel,pwm,velocidad,velocidad\_enc,ref\_pos,posición,posición\_gray,error\_v,int\_v,der\_v,error\_p,int\_p,der\_p,binario"

referencia - valor de referencia del Potenciómetro en voltios

ref\_vel - valor de referencia de velocidad en rpm

pwm - salida del regulador hacia el accionamiento del motor

velocidad - velocidad del eje de baja en rpm

velocidad\_enc - velocidad calculada con el código digital del encoder

ref\_pos - valor de referencia de posición en grados

posición - posición en grados del eje de baja (-180º a +180º)

posición\_gray - posición calculada con el código digital de Gray

error\_v - error a la entrada del regulador de velocidad

int\_v - integral del error de velocidad

der\_v- derivada del error de velocidad

error\_p - error a la entrada del regulador de posición

int\_p - integral del error de posición

der\_p- derivada del error de posición

binario- valor binario de las entradas digitales (Encoder,Gray,Bit de paso por vuelta)

La respuesta al comando “est” es una cadena de caracteres con los parámetros del estado del programa en cada momento:

“Modo\_Control=modocontrol,Potenciometro\_PC=Pot\_PC,modelo=modelo,simulacion=sim,encoder=fb\_e,gray=fb\_g,”

Modo\_Control=modocontrol // Control: vel lazo abierto=1, vel lazo cerrado=2, pos un lazo=3, cascada=4

Potenciometro\_PC=Pot\_PC // Referencia desde el potenciómetro de la unidad Feedback=0 o desde el PC=1

modelo=modelo // Modelo de Feedback original=0, modificado=1

simulacion=sim // Funcionamiento en modo normal=0, simulación=1

encoder=fb\_e // Realimentación de velocidad por tacodinamo=0 o encoder=1

gray=fb\_g // Realimentación de posición por potenciometro=0 o disco Gray=1

# Apéndice: Otros programas para el Arduino

## controlPlantilla

//////////////////////////////////////////////////////////////////

// ########################################################## //

// ########### controlPlantilla ################ //

// ########################################################## //

// //

// Versión 1.0 05/01/2022 11:00 //

// //

// Programa para el control de los módulos de Feedback. //

// //

//////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////

// //

// DENOMINACIÓN DE PINES //

// //

//////////////////////////////////////////////////////////////////

#define PIN\_velocidad A0 //Pin para la señal de velocidad del tacómetro

#define PIN\_posicion A1 //Pin para el potenciómetro de posición

#define PIN\_referencia A2 //Pin para el potenciómetro de referencia

#define PIN\_PWM 13 //Pin para la salida de la señal PWM

//////////////////////////////////////////////////////////////////

// //

// DECLARACIÓN DE VARIABLES //

// //

//////////////////////////////////////////////////////////////////

//Variables para temporización

long int tiempo\_old; // Variable que guarda el último valor de los milisegundos del reloj del Arduino.

long int tiempo\_new; // Variable que guarda el nuevo valor de los milisegundos del reloj del Arduino.

long int tiempo; // Variable en la que se calculan los milisegundos que han pasado.

//ENTRADAS DE LOS SENSORES

int ireferencia; // Entrada del potenciómetro de referencia del equipo Feedback 0...1023

int ivelocidad; // Entrada de la señal de la tacodinamo del equipo Feedback 0...1023

int iposicion; // Entrada de la señal del potenciómetro del equipo Feedback 0...1023

//VARIABLES PARA LA SALIDA PWM

float pwm; // Valor del Duty para la salida PWM como valor real en el rango -127.5...+127.5

int duty; // El Duty de la salida PWM va de 0 (0%) a 255 (100%) (8bits)

// (duty=127...128 => motor parado=50%)

//VALORES DE LAS MAGNITUDES CALCULADAS A TRAVES DE LAS ENTRADAS

float refvelocidad; // Valor interpretado como referencia de velocidad -45...+45 RPM

float velocidad; // Valor estimado de la velocidad en RPM del motor

float refposicion; // Valor interpretado como referencia de posición -180º...+180º

float posicion; // Valor estimado de la posición en grados del eje

//////////////////////////////////////////////////////

// ## DECLARACIÓN DE VARIABLES PARA EL REGULADOR ## //

//////////////////////////////////////////////////////

int Tm=20; // Tiempo de muestreo en ms

float error\_v; // Valor del error de velocidad en el instante k: e\_v(k)

float error\_v\_1; // Valor del error de velocidad en el instante k-1: e\_v(k-1)

float error\_v\_2; // Valor del error de velocidad en el instante k-2: e\_v(k-2)

float error\_p; // Valor del error de posición en el instante k: e\_p(k)

float error\_p\_1; // Valor del error de posición en el instante k-1: e\_p(k-1)

float error\_p\_2; // Valor del error de posición en el instante k-2: e\_p(k-2)

float control; // Valor generado por el regulador para el accionamiento del motor: u(k)

float control\_1; // Valor de "control" en el instante k-1: u(k-1)

float control\_2; // Valor de "control" en el instante k-2: u(k-2)

float Kp; // Ganancia K para reguladores proporcionales

//////////////////////////////////////////////////////////////////

// ############################################################ //

// ######### PROCEDIMIENTO regulador() ######################## //

// ############################################################ //

//////////////////////////////////////////////////////////////////

/\* En esta función se programan los algoritmos de control. \*/

void regulador(){

// #################################################################### //

// En este espacio el estudiantado programará los algoritmos de control

//

// Se dispone de las variables (float):

// refvelocidad: Valor del eje de entrada interpretado como

// referencia de velocidad -45...+45 RPM

// velocidad: Valor estimado de la velocidad en RPM del motor

// a partir de la señal de la tacodinamo

// refposicion: Valor del eje de entrada interpretado como

// referencia de posición -180º...+180º

// posicion: Valor estimado de la posición en grados del eje

// a partir de la señal del potenciómetro

//

// El regulador ha de calcular una señal de "control"

//

// Para ello se facilitan varias variables con las que se pueden

// implementar diferentes algoritmos de control. En algún caso

// es posible que se necesiten declarar y usar otras variables:

//int Tm=20; // Tiempo de muestreo en ms

//float error\_v; // Valor del error de velocidad en el instante k: e\_v(k)

//float error\_v\_1; // Valor del error de velocidad en el instante k-1: e\_v(k-1)

//float error\_v\_2; // Valor del error de velocidad en el instante k-2: e\_v(k-2)

//float error\_p; // Valor del error de posición en el instante k: e\_p(k)

//float error\_p\_1; // Valor del error de posición en el instante k-1: e\_p(k-1)

//float error\_p\_2; // Valor del error de posición en el instante k-2: e\_p(k-2)

//float control; // Valor generado por el regulador para el accionamiento del motor: u(k)

//float control\_1; // Valor de "control" en el instante k-1: u(k-1)

//float control\_2; // Valor de "control" en el instante k-2: u(k-2)

//

// #################################################################### //

}

//// Ejemplo de control en bucle abierto: la señal de "referencia" se pasa directamente a la salida de control

// control=refvelocidad\*5/90; // Paso de -45...+45 RPM a -2.5...+2.5V (0...5V en realidad)

//// Ejemplo de regulador P de velocidad

// Kp=0.2;

// control=Kp\*(refvelocidad-velocidad); // Regulador P=Kp

//// Ejemplo de regulador PI de velocidad

// control=control\_1+(0.18/50)\*(refvelocidad-velocidad); // Regulador I=(9/50)/s discretizado con s=(1-z^-1)/Tm

// control\_1=control; // Se guarda el valor de "control" para el cálculo de la siguiente iteración

//// Ejemplo de regulador P de posición

// Kp=0.2;

// control=Kp\*(refposicion-posicion); // Regulador P=Kp

//////////////////////////////////////////////////////////////////

// //

// PROCEDIMIENTO conversionAD() //

// //

//////////////////////////////////////////////////////////////////

/\* Función mediante la cual se leerán las entradas analógicas 0...5V en un entero 0...1023 (10bits) \*/

void conversionAD(){

ireferencia= analogRead(PIN\_referencia); // Valor entero sin signo 0...1023

iposicion= analogRead(PIN\_posicion); // Valor entero sin signo 0...1023

ivelocidad= analogRead(PIN\_velocidad); // Valor entero sin signo 0...1023

refvelocidad=(float) ireferencia\*90/1023-45; // Paso de 0...1023 a -45...+45 RPM

velocidad=(float) ivelocidad\*230/1023-115; // Paso de 0...1023 a -115...+115 RPM (depende de la taco)

refposicion=(float) ireferencia\*360/1023-180; // Paso de 0...1023 a -180º...+180º

posicion=(float) iposicion\*360/1023-180; // Paso de 0...1023 a -180º...+180º

}

//////////////////////////////////////////////////////////////////

// //

// PROCEDIMIENTO conversionDA() //

// //

//////////////////////////////////////////////////////////////////

/\* Función mediante la cual se convierte -2.5...+2.5V en el duty 0...255 (int, 0%...100%) de la salida PWM \*/

void conversionDA(){

// El duty de la salida PWM va de 0 a 255 (0% a 100%, duty=127...128 => motor parado=50%)

// "pwm" será un valor real (float) calculado por el Regulador escalado entre -127.5 y +127.5 que hay

// que convertir en entero (int) entre 0 y 255 para enviarlo al puerto de salida PWM.

pwm=control\*255/5; // Paso de -2.5...+2.5V a -127.5...+127.5

if (pwm>127.5) pwm=127.5;

if (pwm<-127.5) pwm=-127.5;

// Además, como el motor tiene una zona muerta, se trata de evitar su efecto sumando (giro CW) o

// restando al duty (giro CCW) una cantida aproximadamente equivalente a esa

// zona muerta (+-20) pero manteniendo la linealidad (127.5-20)/127.5.

if (pwm>0) pwm=(pwm\*(127.5-20)/127.5)+20;

else if (pwm<0) pwm=(pwm\*(127.5-20)/127.5)-20;

duty=(int) pwm+127.5; // Paso de float a int y de -127.5...+127.5 a 0...255

// ¡Cuidado con las operaciones con "int" 16bit +-32768 (sin signo:0...65535)!

// Dan lugar a errores por redondeos y desbordamiento.

// Probar:

// duty= ireferencia\*255/1023; (falla cuando ireferencia\*255 > 32768)

// duty= 255\*(ireferencia/1023); (falla porque la división de enteros siempre da 0)

// //if (duty>255) duty=255; // es mejor limitar pwm a 0...255 porque si no puede haber errores al pasarlo a int

// //if (duty<0) duty=0;

analogWrite(PIN\_PWM,duty); // Se envía el valor del duty al puerto de salida PWM

}

//////////////////////////////////////////////////////////////////

// //

// PROCEDIMIENTO enviodatos() //

// //

//////////////////////////////////////////////////////////////////

/\* Envío de datos a traves de la comunicación serie-USB \*/

void enviodatos(){

// Se envía: refvelocidad,velocidad,refposicion,posicion,control,pwm,

//Enviar el dato de REFERENCIA DE VELOCIDAD

Serial.print(refvelocidad);

Serial.print(",");

//Enviar el dato de VELOCIDAD

Serial.print(velocidad);

Serial.print(",");

//Enviar el dato de REFERENCIA DE POSICION

Serial.print(refposicion);

Serial.print(",");

//Enviar el dato de POSICIÓN

Serial.print(posicion);

Serial.print(",");

//Enviar el dato de salida del PID en voltios

Serial.print(control);

Serial.print(",");

//Enviar el dato del duty de la salida PWM corregido a +-127.5

Serial.print(pwm);

Serial.print(",");

//Tiempo transcurrido desde el último envío (debería ser Tm ms, Tiempo de Muestreo)

// tiempo=millis()-tiempo\_new;

// Serial.print(tiempo);

// Serial.print(",");

Serial.print("\n");

}

//////////////////////////////////////////////////////////////////

// ########################################################## //

// ##################### SETUP() ########################### //

// ########################################################## //

//////////////////////////////////////////////////////////////////

void setup() {

Serial.begin(115200); // Velocidad en bits por segundo del puerto serie USB. Resto de parámetros por defecto

//Modo entrada/salida de cada PIN

pinMode(PIN\_posicion, INPUT);

pinMode(PIN\_referencia, INPUT);

pinMode(PIN\_velocidad, INPUT);

pinMode(PIN\_PWM, OUTPUT);

tiempo\_old = millis(); // Inicia el primer valor de los milisegundos del reloj del Arduino.

}

//////////////////////////////////////////////////////////////////

// ########################################################## //

// ###################### LOOP() ########################### //

// ########################################################## //

//////////////////////////////////////////////////////////////////

void loop() { //Bucle de ejecución continua

// Se controla el tiempo que ha pasado desde la anterior ejecución del "control()"

// para ejecutar el algoritmo de control cada Tm ms aproximadamente (Tm: Tiempo de Muestreo).

tiempo\_new = millis(); // Obtiene el nuevo valor de los milisegundos del reloj.

tiempo = tiempo\_new - tiempo\_old; // Calcula los milisegundos que han pasado.

if (tiempo > Tm-1) // "tiempo" debería ser positivo salvo OVERFLOW de las variables que son "long int"

// (no se ha programado el tratamiento del OVERFLOW)

{

conversionAD(); // Lectura de valores de sensores y referencia

regulador(); // Algoritmo de control que genera la señal de control para el accionamiento del motor

conversionDA(); // Envío de la señal de control por la salida PWM hacia el accionamiento del motor

enviodatos(); // Envío de datos por el puerto serie-USB

tiempo\_old=tiempo\_new;

}

}

## controlArduinoComandosBasicos

//////////////////////////////////////////////////////////////////

// ########################################################## //

// ################### controlArduinoComandosBasicos ######## //

// ########################################################## //

// //

// Versión 1.0 05/03/2021 10:00 //

// //

// Programa para el control de los módulos de Feedback a //

// través de instrucciones enviadas al puerto serie USB //

// del ARDUINO //

// //

//////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////

// //

// DENOMINACIÓN DE PINES //

// //

//////////////////////////////////////////////////////////////////

#define PIN\_velocidad A0 //Pin para la señal de velocidad del tacómetro

#define PIN\_posicion A1 //Pin para el potenciómetro de posición

#define PIN\_referencia A2 //Pin para el potenciómetro de referencia

#define PIN\_PWM 13 //Pin para la salida de la señal PWM

//////////////////////////////////////////////////////////////////

// //

// DECLARACIÓN DE VARIABLES //

// //

//////////////////////////////////////////////////////////////////

//Variables para temporización

long int tiempo\_old; // Variable que guarda el anterior valor de los milisegundos del reloj del Arduino.

long int tiempo\_new; // Variable que guarda el nuevo valor de los milisegundos del reloj del Arduino.

long int tiempo; // Variable en la que se calculan los milisegundos que han pasado entre old y new.

//VARIABLES GLOBALES

float referencia; // Potenciómetro de referencia en voltios (0...5V)

float velocidad; // Valor de la velocidad obtenida del tacómetro

float posicion; // Valor de la posición obtenida del potenciómetro

float ref\_pc; // Referencia enviada desde el PC en rpm +-45rpm

//VARIABLES PARA EL REGULADOR

float pwm; // Valor generado por el regulador para la señal PWM

// será un valor que debe limitarse entre +-127.5 (0...255)

int duty; // El Duty de la salida PWM va de 0 (0%) a 255 (100%)

// (duty=127...128 => motor parado=50%)

//VARIABLES PARA RECIBIR DATOS POR EL PUERTO SERIE USB

char inputCommand[12]; //Donde guarda el texto recibido

int inputInt = 0; //Donde guarda el valor Int recibido

float inputFloat = 0.0; //Donde guarda el valor Float recibido

//VARIABLES PARA ELEGIR MODO DE CONTROL, REF, etc.

bool Pot\_PC=0; //Control Potenciómetro=0 o PC=1

//////////////////////////////////////////////////////////////////

// //

// PROCEDIMIENTO leerSerie() //

// //

//////////////////////////////////////////////////////////////////

/\* Esta función lee los caracteres recibidos por el puerto serie (USB)

Espera tres cadenas de caracteres separados por comas:

inputCommand,inputInt,inputFloat

los separa y los convierte al formato correcto de cada valor:

una cadena de caracteres, un valor entero y un valor real.

\*/

void leerSerie() {

const byte buffSize = 32; //Tamaño de la cadena de caracteres

char input[buffSize]; //Cadena de caracteres recibida

byte maxChars = buffSize-1; //31 caracteres de máximo

byte charCont = 0;

byte i = 0;

//Se comprueba si se han recibido datos, si no, no se hace nada

if (Serial.available() > 0) {

input[0] = 0;

delay(1); // Para dar tiempo a que llegue el comando completo (Arduino MEGA, en el MICRO no hace falta)

//Se leen carácter a carácter y se guardanen input

while (Serial.available() > 0) {

if (i > maxChars - 1) {

i = maxChars;

}

input[i] = Serial.read();

i ++;

charCont ++;

}

if (i > maxChars) {

i = maxChars;

}

input[i] = 0;

//DEVOLVER LO QUE ACABA DE RECIBIR PARA COMPROBACION

//

// Serial.print(input);

// Serial.print("\n");

// Serial.flush();

//DATOS de la cadena recibida: inputCommand,inputInt,inputFloat

char \* partirString; //Este puntero indicará donde cortar la cadena

partirString = strtok(input,","); //Separa la primera parte - el texto

strcpy(inputCommand, partirString); //Lo copia en inputCommand

partirString = strtok(NULL, ","); //Separa la segunda parte - el Int

inputInt = atoi(partirString); //Lo convierte a int

partirString = strtok(NULL, ","); //Separa la segunda parte - el Float

inputFloat = atof(partirString); //Lo convierte a float

/\* Se tratan datos recibidos y se opera según el comando recibido.

Comandos aceptados: Todos los comandos pueden ir seguidos de

un "int" y un "float" separados por comas.

dat - Petición de datos del PC. Se enviará el valor de algunas variables.

ppc,int - Selecciona la señal de referencia desde el potenciómetro de la

unidad Feedback (0) o desde el PC (1). P.ej.: 'ppc,1'

ref,int,float - Valor de la referencia desde el PC en el valor "float".

El "int" puede ser 0, p.ej.: 'ref,0,-12.3'

\*/

//PETICION DE DATOS

if (strcmp (inputCommand,"dat") == 0){

envioDatos();

}

//REFERENCIA GENERADA DESDE EL POTENCIOMETRO DEL EQUIPO O DESDE EL PC

else if (strcmp (inputCommand,"ppc") == 0){

Pot\_PC=inputInt;

}

//NUEVO VALOR DE LA REFERENCIA ENVIADA DESDE EL PC

else if (strcmp (inputCommand,"ref") == 0){

ref\_pc=inputFloat;

}

}

}

//////////////////////////////////////////////////////////////////

// //

// PROCEDIMIENTO envioDatos() //

// //

//////////////////////////////////////////////////////////////////

/\* Con esta función se envían los datos por el puerto serie al PC

cada vez que este los pide con el comando "dat".

Los datos van en una cadena separados por comas:

"referencia,ref\_pc,pwm,velocidad,0,0,posicion,0,0,0,0,0,0,0,0,\n"

referencia - valor de referencia del Potenciómetro en voltios +-2.5V (0...5V)

ref\_pc - valor de referencia que se ha recibido del PC en rpm (+-45rpm)

pwm - salida del regulador hacia el accionamiento del motor +-127.5 (0...255)

velocidad - velocidad del eje de baja en rpm

posición - posición en grados del eje de baja +-180º

Se envían en total 15 datos. Varios de esos datos son ceros por compatibildad

con otros programas utilizados en el PC para el control del ARDUINO desde

programación en QT o en MATLAB.

\*/

void envioDatos(){

//Enviar el dato de REFERENCIA leida del potenciómetro

Serial.print(referencia);

Serial.print(",");

//Enviar el dato de REFERENCIA enviada por el PC

Serial.print(ref\_pc);

Serial.print(",");

//Enviar el dato de Control del Motor (salida del Reulador PID si se implementa)

Serial.print(pwm);

Serial.print(",");

//Enviar el dato de VELOCIDAD

Serial.print(velocidad);

Serial.print(",");

//Relleno

Serial.print("0,0,");

//Enviar el dato de POSICIÓN

Serial.print(posicion);

Serial.print(",");

//Relleno

Serial.print("0,0,0,0,0,0,0,0,");

Serial.print("\n");

// Enviar todo

Serial.flush();

}

//////////////////////////////////////////////////////////////////

// //

// PROCEDIMIENTO conversionAD() //

// //

//////////////////////////////////////////////////////////////////

/\* Función mediante la cual se leerán las entradas analógicas. \*/

void conversionAD(){

referencia=(float) analogRead(PIN\_referencia)\*5/1023; // Paso a voltios 0...5V

posicion=(float) analogRead(PIN\_posicion)\*360/1023-180; // Paso a grados +-180º

velocidad=(float) analogRead(PIN\_velocidad)\*225/1023-112.5; // Paso a rpm aproximadamente +-112.5rmp

}

//////////////////////////////////////////////////////////////////

// //

// PROCEDIMIENTO control() //

// //

//////////////////////////////////////////////////////////////////

/\* En esta función se recogen los algoritmos de control. \*/

void control(){

if (Pot\_PC) { // Si Pot\_PC=1 referencia enviada desde el PC

pwm=ref\_pc\*255/90; // Paso de referencia +-45rpm a pwm +-127.5 (0...255)

}

else { // Si Pot\_PC=0 referencia del potenciómetro del Feedback

pwm=referencia\*255/5-127.5; // Paso de referencia 0...5V a pwm +-127.5 (0...255)

}

if (pwm>127.5) pwm=127.5; // Limitar el PWM a +-127.5

if (pwm<-127.5) pwm=-127.5;

duty=(int) (pwm+127.5); // El duty va de 0 a 255 (0% a 100%, duty=127...128 => motor parado=50%)

analogWrite(PIN\_PWM,duty); // Se envía el valor del duty al motor por el puerto se salida PWM

}

//////////////////////////////////////////////////////////////////

// ########################################################## //

// ##################### SETUP() ########################### //

// ########################################################## //

//////////////////////////////////////////////////////////////////

void setup() {

Serial.begin(115200); // Velocidad en bits por segundo del puerto serie USB. Resto de parámetros por defecto

//Modo entrada/salida de cada PIN

pinMode(PIN\_posicion, INPUT);

pinMode(PIN\_referencia, INPUT);

pinMode(PIN\_velocidad, INPUT);

pinMode(PIN\_PWM, OUTPUT);

tiempo\_old = millis(); // Inicia el primer valor de los milisegundos del reloj del Arduino.

}

//////////////////////////////////////////////////////////////////

// ########################################################## //

// ###################### LOOP() ########################### //

// ########################################################## //

//////////////////////////////////////////////////////////////////

void loop() { //Bucle de ejecución continua

leerSerie(); // Comprobar si han llegado órdenes por el puerto serie

// Se controla el tiempo que ha pasado desde la anterior ejecución del "control()"

// para ejecutar el algoritmo de control cada 20ms aproximadamente.

tiempo\_new = millis(); // Obtiene el nuevo valor de los milisegundos del reloj.

tiempo = tiempo\_new - tiempo\_old; // Calcula los milisegundos que han pasado.

if (tiempo > 19) // "tiempo" debería ser positivo salvo OVERFLOW de las variables que son "long int"

{ // (no se ha programado el tratamiento del OVERFLOW, tardaría un mes aprox.)

conversionAD(); // Lectura de valores de sensores y referencia

control(); // Algoritmo de control que genera la salida PWM

tiempo\_old=tiempo\_new;

}

}

# Apéndice: Programas en MATLAB para calibración

## identificaFB

% Script "identificaFB"

%

% J.A. Sirgo 13/01/2022

%

% Realiza la identificación de un módulo de FeedBack mediante la conexión

% del ARDUINO del módulo a un puerto USB del ordenador.

%

% Proporciona la FdT del motor entre la entrada PWM (-2.5V a +2.5V) y la

% velocidad del eje de salida de la unidad mecánica (eje de baja) en rpm.

%

% El script genera varias variables entre las que destacan:

%

% G1: FdT de primer orden del sistema identificado.

% K: Ganancia del sistema

% T: Constante de tiempo del sistema

% G2: FdT de segundo orden del sistema identificado.

% ENSAYO: Es una matriz con las mismas columnas que se guardan en el

% fichero de datos.

%

% Columna 1 Columna 2 Columna 3 Columna 4 Columna 5

%

% Tiempo Referencia PWM Velocidad Vel\_encoder

% (segundos) (rpm) (+-2.5V) (rpm) (rpm)

%

% Se guardan los datos del ensayo en un fichero de texto con las cinco

% columnas de la matriz ENSAYO y cuyo nombre es la fecha y hora del ensayo.

%

% Se visualizan al final unas gráficas de las variables durante el ensayo

% realizado para la identificación y la comparación de la respuesta a un

% escalón unitario de G1 y G2.

% PUESTA A CERO:

%===============

clear R; % Almacena el valor de las Referencias (rpm)

clear U; % Almacena el valor de las Entradas (PWM +-127)

clear Y; % Almacena el valor de las Salidas (Velocidad en rpm)

clear V; % Almacena el valor de la velocidad medida por el encoder en rpm

clear T; % Almacena el tiempo transcurrido (segundos)

r=0; % referencia (rpm)

u=0; % entrada (PWM)

y=0; % salida (rpm)

v=0; % velocidad medida por el encoder (rpm)

t=0; % tiempo (segundos)

t\_ensayo=12; % Tiempo del ensayo en segundos

k=1; % Ã­ndice de los vectores de datos

cambio=0; % 0 escalón positivo, 1 escalón negativo

% SETUP:

%=======

% Configuración del puerto serie USB donde está conectado el ARDUINO

fprintf('Configuración del puerto serie\n');

clear serialUSB; % Libera el handler 'serialUSB' del puerto serie si estaba usado

newobjs = instrfind; % Obtiene los objetos serie activos

if length(newobjs)~=0 fclose(newobjs); end % Cierra los objetos serie que hayan quedado abiertos

fprintf('Puertos serie ocupados liberados\n');

% Para ver los puertos en uso, desde una ventana de comandos de Windows

% se puede ejecutar C:\> mode ( >>!mode )

[status,cmdout] = system('mode'); % cmdout tiene la salida por pantalla del comando

pos=strfind(cmdout,'COM'); % Posición de los 'COM' en el texto, COM1 y el COM del ARDUINO

port=cmdout(pos(length(pos)):pos(length(pos))+3); % Se extrae la cadena del segundo (último) 'COM', habitualmente el USB

% port=cmdout(pos(length(pos)):pos(length(pos))+4); % +4 si el puerto es COM10 o superior (C+4 caracteres)

% del ARDUINO en los PCs del laboratorio, si no hay más dispositivos conectados

% por puerto serie y teniendo en cuenta que los PCs del laboratorio tienen un puerto serie

% COM1 no usado.

fprintf('Usando puerto serie: "%s"\n',port);

serialUSB = serial(port,'BaudRate',115200, 'Terminator',44); % Se configura el puesto serie.

% serialUSB.Terminator=44; % Terminador ',' (ascii 44) para que al leer las líneas se reconozca cada dato que envía el ARDUINO

serialUSB.timeout=1;

fopen(serialUSB); % Abre el puerto serie: 'COM3', 'COM4', 'COM5', etc.

% pause(1); % Necesario para que el puerto USB del Arduino MEGA (chino) esté listo

fprintf('Puerto serie configurado y abierto\n');

fprintf(serialUSB,'rst,0,0'); % Comando 'rst': el ARDUINO se pone en los valores por defecto.

% pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

% fprintf(serialUSB,'sim,1,0'); % Comando 'sim': el ARDUINO se pone en modo simulación (para pruebas).

% pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

fprintf(serialUSB,'ppc,1,0'); % Comando 'ppc': el ARDUINO acepta los valores de referencia de velocidad desde el PC

% pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

fprintf(serialUSB,'con,1,0'); % Comando 'con': el ARDUINO pone el control de velocidad en lazo abierto

% La referencia de velocidad será en rpm

% estimadas en lazo abierto.

% pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

fprintf(serialUSB,'ref,0,%f',r); % Comando 'ref': Indica el valor de referencia en rpm.

R(k) = r; % Primera línea de los vectores de datos a cero.

U(k) = u;

Y(k) = y;

T(k) = t;

% ENSAYO:

%========

% Un escalón +40 cuatro segundos, escalón -80 (referencia a -40)

% cuatro segundos y referencia a 0 dos segundos.

tic; % Inicio de la cuenta de tiempo t=0

t=toc; % toc es el tiempo transcurrido en segundos hasta este instante desde el tic

t\_old=t; % Tiempo actual para comparar con el tiempo transcurrido

t\_change=2; % Tiempo hasta un nuevo cambio de referencia.

fprintf('Tiempo transcurrido: %3.0f de %3.0f segundos',t,t\_ensayo);

while t<t\_ensayo-0.1 % Tiempo del ensayo

k = k + 1;

% pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

% Valores actuales de entradas y salidas

fprintf(serialUSB,'dat,0,0'); % Comando 'dat' para que el ARDUINO devuelva sus datos

for i=1:15

datos(i) = str2double(fscanf(serialUSB,'%s',44)); % Hay 15 datos cada dato se lee como un double

end

% Los datos recibidos son:

%

% referencia ref\_vel pwm velocidad vel\_enc ref\_pos posición pos\_gray error\_v int\_v der\_v error\_p int\_p der\_p binario

%

% r=datos(2);

u=datos(3); % Entrada PWM

y=datos(4); % Salida Velocidad

v=datos(5); % velocidad medida por el encoder

t=toc; % Tiempo transcurrido hasta ahora.

t\_new=t; % Tiempo en este instante.

% Escalón positivo: referencia +40, 4 segundos

if ((t\_new-t\_old)>t\_change && cambio==0) % Si se ha cumplido el tiempo hasta el nuevo cambio de referencia

r=40;

t\_change=4;

cambio=1;

% Envío de la nueva referencia:

fprintf(serialUSB,'ref,0,%f',r);

t\_old=t\_new; % Se actualiza el tiempo para el siguiente cambio de referencia.

fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\bTiempo transcurrido: %3.0f de %3.0f segundos',t,t\_ensayo);

end

% Escalón negativo: referencia -40, 4 segundos

if ((t\_new-t\_old)>t\_change && cambio==1) % Si se ha cumplido el tiempo hasta el nuevo cambio de referencia

r=-40;

t\_change=4;

cambio=2;

% Envío de la nueva referencia:

fprintf(serialUSB,'ref,0,%f',r);

t\_old=t\_new; % Se actualiza el tiempo para el siguiente cambio de referencia.

fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\bTiempo transcurrido: %3.0f de %3.0f segundos',t,t\_ensayo);

end

% Parada: referencia 0, 2 segundos

if ((t\_new-t\_old)>t\_change && cambio==2) % Si se ha cumplido el tiempo hasta el nuevo cambio de referencia

r=0;

t\_change=2;

cambio=0;

% Envío de la nueva referencia:

fprintf(serialUSB,'ref,0,%f',r);

t\_old=t\_new; % Se actualiza el tiempo para el siguiente cambio de referencia.

fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\bTiempo transcurrido: %3.0f de %3.0f segundos',t,t\_ensayo);

end

% Se guardan los datos de todas las variables en vectores

R(k) = r;

U(k) = u;

Y(k) = y;

V(k) = v;

T(k) = t;

end

fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\bTiempo transcurrido: %3.0f de %3.0f segundos',t,t\_ensayo);

% ENSAYO es una matriz que contendrá todos los datos del ensayo en cinco

% columnas como las del fichero en el que se guardan los datos del ensayo.

ENSAYO=[T;R;U;Y;V]'; % Resultados del ensayo

% PUESTA A CERO FINAL

%====================

r=0; % Referencia a cero

fprintf(serialUSB,'ref,0,%f',r);

fprintf(serialUSB,'rst,0,0'); % Comando 'rst': el ARDUINO se pone en los valores por defecto.

fclose(serialUSB);

delete(serialUSB);

clear serialUSB; % Libera el handler 'serialUSB' del puerto serie USB

% GRÁFICA DEL ENSAYO

%===================

subplot(2,1,1);

plot(T,[U;Y]); % PWM/Velocidad(rpm) (o plot(T,[R;U;Y]); Referencia/PWM/Velocidad)

title('Resultado del experimento')

legend('Entrada V', 'Salida RPM')

% FICHERO DE DATOS DEL ENSAYO:

%=============================

% Es fichero de texto de cinco columnas cuyo nombre son la fecha y hora

% del ensayo:

%

% Columna 1 Columna 2 Columna 3 Columna 4 Columna 5

%

% Tiempo Referencia PWM Velocidad Vel\_encoder

% (segundos) (rpm) (+-2.5V) (rpm) (rpm)

nombre=['identificaFB\_',datestr(now,30),'.txt']; % El nombre es la fecha y hora actual

f\_out=fopen(nombre, 'w'); % Abre para escritura 'w'

for i=1:length(ENSAYO) % Guardar datos en el fichero

fprintf(f\_out, '%f %f %f %f %f \n' ,ENSAYO(i,:));

end;

fclose(f\_out); % Cerrar el fichero de escritura

% IDENTIFICACIÓN

%===============

% Los datos se obtienen a intervalos de aproximadamente 30ms (dependerá de

% múltiples factores) pero no están muestreados de manera uniforme. Se

% remuestrean con un tiempo de muestreo Tm fijo.

ds=timeseries(ENSAYO(:,3:4),ENSAYO(:,1)); % Objeto timeseries para remuestrear los datos

Tm=0.01; % Tiempo de muestreo elegido en segundos

tiempo=[0:Tm:11.80]; % Nuevo vector de tiempo de 0 a 11.8 segundos

ds=resample(ds,tiempo); % Remuestrea los datos a un Tm uniforme

datos\_id=iddata(ds.Data(:,2),ds.Data(:,1),Tm); % iddata( Salida Velocidad(rpm) , Entrada PWM , Tiempo de muestreo)

sys1=idproc('P1'); % Estructura para identrificar un sistema de primer orden

sys2=idproc('P2'); % Estructura para identrificar un sistema de segundo orden

Gest2=pem(datos\_id,sys2); % Identificación del sistema de segundo orden

Gest1=pem(datos\_id,sys1); % Identificación del sistema de primer orden

% RESULTADOS POR PANTALLA

%========================

s=tf('s');

fprintf('\n \nFunción de Transferencia de segundo orden:')

G2=Gest2.K.value/(Gest2.Tp1.value\*s+1)/(Gest2.Tp2.value\*s+1) % FdT segundo orden

[pole,zero]=pzmap(G2);

fprintf('K=%f p1=%f p2=%f ',Gest2.K.value,pole(2),pole(1))

fprintf('\n \nFunción de Transferencia de primer orden:')

G1=Gest1.K.value/(Gest1.Tp1.value\*s+1) % FdT primer orden

K=Gest1.K.value ; % Ganancia K del sistema de primer orden

T=Gest1.Tp1.value ; % Constante de tiempo T del sistema de primer orden

fprintf('K=%f T=%f p1=%f \n',K,T,-1/T)

% COMPARACION G1 G2 A LA RESPUESTA ESCALÓN

%=========================================

subplot(2,1,2);

step(G1,G2);

title('Resultado del experimento')

legend('Entrada V', 'Salida RPM')

## identifica

% Script 'identifica'

%

% J.A. Sirgo 18/03/2021

%

% Realiza la identificación de un módulo de FeedBack mediante la conexión

% del ARDUINO del módulo a un puerto USB del ordenador.

%

% Proporciona la FdT del motor entre la entrada PWM (-2.5 a +2.5) y la

% velocidad del eje de salida de la unidad mecánica (eje de baja) en rpm.

%

% El script genera varias variables entre las que destacan:

%

% G1: FdT de primer orden del sistema identificado.

% K: Ganancia del sistema

% T: Constante de tiempo del sistema

% G2: FdT de segundo orden del sistema identificado.

% ENSAYO: Es una matriz con las mismas columnas que se guardan en el

% fichero de datos.

%

% Columna 1 Columna 2 Columna 3 Columna 4 Columna 5

%

% Tiempo Referencia PWM Velocidad Vel\_encoder

% (segundos) (rpm) (+-2.5V) (rpm) (rpm)

%

% Se guardan los datos del ensayo en un fichero de texto con las cinco

% columnas de la matriz ENSAYO y cuyo nombre es la fecha y hora del ensayo.

%

% Los datos se obtienen a intervalos de aproximadamente 30ms (dependerá de

% múltiples factores) pero no están muestreados de manera uniforme. Se

% remuestrean con un tiempo de muestreo Tm fijo y quedan en las variables:

%

% ds: Son los datos de entrada (PWM) y salida (Velocidad) de la matriz

% ENSAYO remuestreados con un tiempo de muestreo fijo Tm.

%

% ts: Tiene los valores del tiempo correspondientes a la matriz ds

% remuestreados. El valor de ts(2) es el tiempo de muestreo Tm.

%

% Se visualizan al final unas gráficas de las variables durante el ensayo

% realizado para la identificación.

%

% El script tiene comentadas unas líneas que permiten generar gráficas en

% tiempo real con los valores de las variables, pero no es recomendable

% utilizarlas porque ralentizan el muestreo (aproximadamente 100ms) y

% falsean los datos de la identificación final del sistema.

% PUESTA A CERO:

%===============

clear R; % Almacena el valor de las Referencias (rpm)

clear U; % Almacena el valor de las Entradas (PWM +-127)

clear Y; % Almacena el valor de las Salidas (Velocidad en rpm)

clear V; % Almacena el valor de la velocidad medida por el encoder en rpm

clear T; % Almacena el tiempo transcurrido (segundos)

clear P; % Almacena los valores representados en las Gráficas en tiempo real

% Valores iniciales:

r=0; % referencia (rpm)

u=0; % entrada (PWM)

y=0; % salida (rpm)

v=0; % velocidad medida por el encoder (rpm)

t=0; % tiempo (segundos)

t\_ensayo=60; % Tiempo del ensayo en segundos (mínimo 1 minuto, recomendable 5 a 10 minutos)

k=1; % índice de los vectores de datos

l = 200; % Salto de las 200 primeras muestras (para los gráficos en tiempo real)

% SETUP:

%=======

% Configuración del puerto serie USB conectado con el ARDUINO

clear serialUSB; % Libera el handler 's' del puerto serie si estaba usado

newobjs = instrfind; % Obtiene los objetos serie activos

if length(newobjs)~=0 fclose(newobjs); end % Cierra los objetos serie que hayan quedado abiertos

% % Instrucciones que solo funcionan en versiones modernas de MATLAB R2020

% listaUSB=serialportlist; % Lista de dispositivos serie conectados 'a los puertos USB'

% serialUSB = serialport(listaUSB(1),115200); % Abre el puerto serie (1): 'COM3', 'COM5', etc.

%(habitualmente el USB del ARDUINO, si no hay más dispositivos conectados por puerto serie)

% Para ver los puertos, desde una ventana de comandos de Windows

% se puede ejecutar C:\> mode ( >>!mode )

% Usando serial(), fopen(), fprintf(), etc. aparentemente la comunicación es

% más rápida y el programa compatible con versiones de MATLAB anteriores a

% R2020, aunque serial() NO SE RECOMIENDA desde R2019.

% Por lo tanto se sustituyen los comandos comentados en las líneas

% anteriores por:

% Para ver los puertos en uso, desde una ventana de comandos de Windows

% se puede ejecutar C:\> mode ( >>!mode )

[status,cmdout] = system('mode'); % cmdout tiene la salida por pantalla del comando

pos=strfind(cmdout,'COM'); % Posición de los 'COM' en el texto, COM1 y el COM del ARDUINO

port=cmdout(pos(length(pos)):pos(length(pos))+3); % Se extrae la cadena del segundo (último) 'COM', habitualmente el USB

% port=cmdout(pos(length(pos)):pos(length(pos))+4); % +4 si el puerto es COM10 o superior (C+4 caracteres)

% del ARDUINO en los PCs del laboratorio, si no hay más dispositivos conectados

% por puerto serie y teniendo en cuenta que los PCs del laboratorio tienen un puerto serie

% COM1 no usado.

serialUSB = serial(port,'BaudRate',115200, 'Terminator',44);

% Terminador ',' (ascii 44) para que al leer las líneas se reconozca cada dato que envía el ARDUINO

serialUSB.timeout=1;

fopen(serialUSB);

% pause(1); % Necesario para que el puerto USB del Arduino MEGA (chino) esté listo

% % Instrucciones que solo funcionan en versiones modernas de MATLAB R2020

% configureTerminator(serialUSB,0); % Terminador que se envía tras cada writeline() para que el ARDUINO

% writeline(serialUSB,'rst,0,0'); % Comando 'rst': el ARDUINO se pone en los valores por defecto.

% writeline(serialUSB,'con,1,0'); % Comando 'con': el ARDUINO se pone en modo control de velocidad.

% writeline(serialUSB,'ppc,1,0'); % Comando 'ppc': el ARDUINO acepta los valores de referencia de velocidad desde el PC

% % La referencia de velocidad será en rpm

% % estimadas en lazo abierto.

% str\_ref=compose('ref,0,%f',r); % Comando 'ref': Indica el valor de referencia en rpm.

% writeline(serialUSB,str\_ref);

fprintf(serialUSB,'rst,0,0'); % Comando 'rst': el ARDUINO se pone en los valores por defecto.

% pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

% fprintf(serialUSB,'sim,1,0'); % Comando 'sim': el ARDUINO se pone en modo simulación (para pruebas).

% pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

fprintf(serialUSB,'ppc,1,0'); % Comando 'ppc': el ARDUINO acepta los valores de referencia de velocidad desde el PC

% pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

fprintf(serialUSB,'con,1,0'); % Comando 'con': el ARDUINO pone el control de velocidad en lazo abierto

% La referencia de velocidad será en rpm

% estimadas en lazo abierto.

% pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

fprintf(serialUSB,'ref,0,%f',r); % Comando 'ref': Indica el valor de referencia en rpm.

% Tm = 1; % Tiempo de muestreo

R(k) = r; % Primera línea de los vectores de datos a cero.

U(k) = u;

Y(k) = y;

T(k) = t;

% ENSAYO:

%========

tic; % Inicio de la cuenta de tiempo t=0

t=toc; % toc es el tiempo transcurrido en segundos hasta este instante desde el tic

t\_old=t; % Tiempo actual para comparar con el tiempo transcurrido

t\_change=1; % Tiempo aleatorio hasta un nuevo cambio de referencia.

fprintf('Tiempo transcurrido: %3.0f de %3.0f segundos',t,t\_ensayo);

% Bucle del ensayo

while t<t\_ensayo % Tiempo del ensayo en segundos (mínimo 1 minuto, recomendable 5 a 10 minutos)

k = k + 1;

l = l + 1;

% pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

fprintf(serialUSB,'dat,0,0'); % Comando 'dat' para que el ARDUINO devuelva sus datos

% serialUSB.Terminator=44; % Terminador ',' (ascii 44) para que al leer las líneas se reconozca cada dato que envía el ARDUINO

for i=1:15

datos(i) = str2double(fscanf(serialUSB,'%s',44)); % Hay 15 datos cada dato se lee como un double

end

% Los datos recibidos son:

%

% referencia ref\_vel pwm velocidad vel\_enc ref\_pos posición pos\_gray error\_v int\_v der\_v error\_p int\_p der\_p binario

%

% r=datos(2);

u=datos(3); % Entrada PWM

y=datos(4); % Salida Velocidad

v=datos(5); % velocidad medida por el encoder

%t = k\*Tm;

t=toc; % Tiempo transcurrido hasta ahora.

t\_new=t; % Tiempo en este instante.

if (t\_new-t\_old)>t\_change % Si se ha cumplido el tiempo hasta el nuevo cambio de referencia

r=fix(80\*rand()-40); % Nueva referencia, un entero aleatorio entre -40 y +40 rpm

t\_change=5\*rand()+0.2; % Tiempo hasta el nuevo cambio de referencia entre 0.2 a 5.2 segundos

% Envío de la nueva referencia:

% str\_ref=compose("ref,0,%f",r);

% writeline(serialUSB,str\_ref);

fprintf(serialUSB,'ref,0,%f',r);

t\_old=t\_new; % Se actualiza el tiempo para el siguiente cambio de referencia.

fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\bTiempo transcurrido: %3.0f de %3.0f segundos',t,t\_ensayo);

end

% Se guardan los datos de todas las variables en vectores

R(k) = r;

U(k) = u;

Y(k) = y;

V(k) = v;

T(k) = t;

% % DIBUJO DE LA GRÁFICA EN TIEMP REAL:

% %====================================

% % El script tiene comentadas estas líneas que permiten generar gráficas en

% % tiempo real con los valores de las variables, pero no es recomendable

% % utilizarlas porque ralentizan el muestreo (aproximadamente 100ms) y falsean

% % los datos de la identificación final del sistema.

%

% figure(1);

%

% P(:,l) = [r;u;y];

%

% etiq = {'referencia','PWM','velocidad'}; for j = 1:3,

% subplot(3,1,j); if l<200

% plot(P(j,:));

% else

% plot(P(j,l-199:l));

% end

% axis([0 200 -130 130]); title(etiq{j});

% end

%

% drawnow;

end

% ENSAYO es una matriz que contendrá todos los datos del ensayo en cinco

% columnas como las del fichero en el que se guardan los datos del ensayo.

ENSAYO=[T;R;U;Y;V]'; % Resultados del ensayo

% PUESTA A CERO FINAL

%====================

r=0; % Referencia a cero

% configureTerminator(serialUSB,0);

% str\_ref=compose("ref,0,%f",r); % Comando 'ref': Indica el valor de referencia en rpm.

% writeline(serialUSB,str\_ref);

% writeline(serialUSB,'rst'); % Comando 'rst': el ARDUINO se pone en los valores por defecto.

fprintf(serialUSB,'ref,0,%f',r); % Comando 'ref': Indica el valor de referencia en rpm.

fprintf(serialUSB,'rst,0,0'); % Comando 'rst': el ARDUINO se pone en los valores por defecto.

fclose(serialUSB);

delete(serialUSB);

clear serialUSB; % Libera el handler 's' del puerto serie USB

% GRÁFICA DEL ENSAYO

%===================

figure(2);

% subplot(2,1,1);

plot(T,[U;Y]); % PWM/Velocidad(rpm) (o plot(T,[R;U;Y]); Referencia/PWM/Velocidad)

% FICHERO DE DATOS DEL ENSAYO:

%=============================

% Es fichero de texto de cinco columnas cuyo nombre son la fecha y hora

% del ensayo:

%

% Columna 1 Columna 2 Columna 3 Columna 4 Columna 5

%

% Tiempo Referencia PWM Velocidad Vel\_encoder

% (segundos) (rpm) (+-127) (rpm) (rpm)

% nombre=['identifica\_',datestr(now,30),'.txt']; % El nombre es la fecha y hora actual

% f\_out=fopen(nombre, 'w'); % Abre para escritura 'w'

% for i=1:length(ENSAYO) % Guardar datos en el fichero

% fprintf(f\_out, '%f %f %f %f %f \n' ,ENSAYO(i,:));

% end;

% fclose(f\_out); % Cerrar el fichero de escritura

% IDENTIFICACIÓN para versiones de MATLAB posteriores a R2015

%============================================================

% Los datos se obtienen a intervalos de aproximadamente 30ms (dependerá de

% múltiples factores) pero no están muestreados de manera uniforme. Se

% remuestrean con un tiempo de muestreo Tm fijo.

% [ds,ts]=resample(ENSAYO(:,[3;4]),ENSAYO(:,1)); % Remuestrea los datos a un Tm uniforme

% datos\_id=iddata(ds(:,2),ds(:,1),ts(2)); % ts(2) es el Tm

%

% Gest1=tfest(datos\_id,1,0); % Estima una FdT de primer orden sin ceros

% Gest2=tfest(datos\_id,2,0); % Estima una FdT de segundo orden sin ceros

% RESULTADOS POR PANTALLA

%========================

% % fprintf('\n \nFunción de Transferencia de segundo orden:')

% % G2=tf(Gest2.numerator,Gest2.denominator) % FdT obtenida

% % [pole,zero]=pzmap(G2);

% % fprintf('K=%f p1=%f p2=%f ',Gest2.numerator/Gest2.denominator(3),pole(2),pole(1))

% fprintf('\n \nFunción de Transferencia de primer orden:')

% G1=tf(Gest1.numerator,Gest1.denominator) % FdT obtenida

% K=Gest1.numerator/Gest1.denominator(2); % Ganancia

% T=1/Gest1.denominator(2); % Constante de tiempo

% fprintf('K=%f T=%f p1=%f \n',K,T,-1/T)

% % IDENTIFICACION para versiones anteriores de MATLAB

% %===================================================

ds=timeseries(ENSAYO(:,3:4),ENSAYO(:,1)); % Objeto timeseries para remuestrear los datos

Tm=0.01; % Tiempo de muestreo elegido en segundos

tiempo=[0:Tm:11.80]; % Nuevo vector de tiempo de 0 a 11.8 segundos

ds=resample(ds,tiempo); % Remuestrea los datos a un Tm uniforme

datos\_id=iddata(ds.Data(:,2),ds.Data(:,1),Tm); % iddata( Salida Velocidad(rpm) , Entrada PWM , Tiempo de muestreo)

sys1=idproc('P1'); % Estructura para identrificar un sistema de primer orden

sys2=idproc('P2'); % Estructura para identrificar un sistema de segundo orden

Gest2=pem(datos\_id,sys2); % Identificación del sistema de segundo orden

Gest1=pem(datos\_id,sys1); % Identificación del sistema de primer orden

% RESULTADOS POR PANTALLA

%========================

s=tf('s');

% fprintf('\n \nFunción de Transferencia de segundo orden:')

% G2=Gest2.K.value/(Gest2.Tp1.value\*s+1)/(Gest2.Tp2.value\*s+1) % FdT segundo orden

% [pole,zero]=pzmap(G2);

% fprintf('K=%f p1=%f p2=%f ',Gest2.K.value,pole(2),pole(1))

fprintf('\n \nFunción de Transferencia de primer orden:')

G1=Gest1.K.value/(Gest1.Tp1.value\*s+1) % FdT primer orden

K=Gest1.K.value ; % Ganancia K del sistema de primer orden

T=Gest1.Tp1.value ; % Constante de tiempo T del sistema de primer orden

fprintf('K=%f T=%f p1=%f \n',K,T,-1/T)

%

% % COMPARACION G1 G2 A LA RESPUESTA ESCALÓN

% %=========================================

% subplot(2,1,2);

% step(G1,G2);

## tunningFB

% Script "tunningFB"

%

% J.A. Sirgo 13/01/2022

%

% Realiza la identificación de un módulo de FeedBack mediante la conexión

% del ARDUINO del módulo a un puerto USB del ordenador.

%

% Proporciona la FdT del motor entre la entrada PWM (-2.5V a +2.5V) y la

% velocidad del eje de salida de la unidad mecánica (eje de baja) en rpm.

%

% El script genera varias variables entre las que destacan:

%

% G1: FdT de primer orden del sistema identificado.

% K: Ganancia del sistema

% T: Constante de tiempo del sistema

% G2: FdT de segundo orden del sistema identificado.

% ENSAYO: Es una matriz con las mismas columnas que se guardan en el

% fichero de datos.

%

% Columna 1 Columna 2 Columna 3 Columna 4 Columna 5

%

% Tiempo Referencia PWM Velocidad Vel\_encoder

% (segundos) (rpm) (+-2.5V) (rpm) (rpm)

%

% Se guardan los datos del ensayo en un fichero de texto con las cinco

% columnas de la matriz ENSAYO y cuyo nombre es la fecha y hora del ensayo.

%

% Se visualizan al final unas gráficas de las variables durante el ensayo

% realizado para la identificación y la comparación de la respuesta a un

% escalón unitario de G1 y G2.

% PUESTA A CERO:

%===============

clear R; % Almacena el valor de las Referencias (rpm)

clear U; % Almacena el valor de las Entradas (PWM +-127)

clear Y; % Almacena el valor de las Salidas (Velocidad en rpm)

clear V; % Almacena el valor de la velocidad medida por el encoder en rpm

clear T; % Almacena el tiempo transcurrido (segundos)

r=0; % referencia (rpm)

u=0; % entrada (PWM)

y=0; % salida (rpm)

v=0; % velocidad medida por el encoder (rpm)

t=0; % tiempo (segundos)

t\_ensayo=12; % Tiempo del ensayo en segundos

k=1; % Ã­ndice de los vectores de datos

cambio=0; % 0 escalón positivo, 1 escalón negativo

% SETUP:

%=======

% Configuración del puerto serie USB donde está conectado el ARDUINO

fprintf('Configuración del puerto serie\n');

clear serialUSB; % Libera el handler 'serialUSB' del puerto serie si estaba usado

newobjs = instrfind; % Obtiene los objetos serie activos

if length(newobjs)~=0 fclose(newobjs); end % Cierra los objetos serie que hayan quedado abiertos

fprintf('Puertos serie ocupados liberados\n');

% Para ver los puertos en uso, desde una ventana de comandos de Windows

% se puede ejecutar C:\> mode ( >>!mode )

[status,cmdout] = system('mode'); % cmdout tiene la salida por pantalla del comando

pos=strfind(cmdout,'COM'); % Posición de los 'COM' en el texto, COM1 y el COM del ARDUINO

port=cmdout(pos(length(pos)):pos(length(pos))+3); % Se extrae la cadena del segundo (último) 'COM', habitualmente el USB

% port=cmdout(pos(length(pos)):pos(length(pos))+4); % +4 si el puerto es COM10 o superior (C+4 caracteres)

% del ARDUINO en los PCs del laboratorio, si no hay más dispositivos conectados

% por puerto serie y teniendo en cuenta que los PCs del laboratorio tienen un puerto serie

% COM1 no usado.

fprintf('Usando puerto serie: "%s"\n',port);

serialUSB = serial(port,'BaudRate',115200, 'Terminator',44); % Se configura el puesto serie.

% serialUSB.Terminator=44; % Terminador ',' (ascii 44) para que al leer las líneas se reconozca cada dato que envía el ARDUINO

serialUSB.timeout=1;

fopen(serialUSB); % Abre el puerto serie: 'COM3', 'COM4', 'COM5', etc.

% pause(1); % Necesario para que el puerto USB del Arduino MEGA (chino) esté listo

fprintf('Puerto serie configurado y abierto\n');

fprintf(serialUSB,'rst,0,0'); % Comando 'rst': el ARDUINO se pone en los valores por defecto.

% pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

% fprintf(serialUSB,'sim,1,0'); % Comando 'sim': el ARDUINO se pone en modo simulación (para pruebas).

% pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

fprintf(serialUSB,'ppc,1,0'); % Comando 'ppc': el ARDUINO acepta los valores de referencia de velocidad desde el PC

% pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

fprintf(serialUSB,'con,1,0'); % Comando 'con': el ARDUINO pone el control de velocidad en lazo abierto

% La referencia de velocidad será en rpm

% estimadas en lazo abierto.

% pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

fprintf(serialUSB,'ref,0,%f',r); % Comando 'ref': Indica el valor de referencia en rpm.

R(k) = r; % Primera línea de los vectores de datos a cero.

U(k) = u;

Y(k) = y;

T(k) = t;

% ENSAYO:

%========

% Un escalón +40 cuatro segundos, escalón -80 (referencia a -40)

% cuatro segundos y referencia a 0 dos segundos.

tic; % Inicio de la cuenta de tiempo t=0

t=toc; % toc es el tiempo transcurrido en segundos hasta este instante desde el tic

t\_old=t; % Tiempo actual para comparar con el tiempo transcurrido

t\_change=2; % Tiempo hasta un nuevo cambio de referencia.

fprintf('Tiempo transcurrido: %3.0f de %3.0f segundos',t,t\_ensayo);

while t<t\_ensayo-0.1 % Tiempo del ensayo

k = k + 1;

% pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

% Valores actuales de entradas y salidas

fprintf(serialUSB,'dat,0,0'); % Comando 'dat' para que el ARDUINO devuelva sus datos

for i=1:15

datos(i) = str2double(fscanf(serialUSB,'%s',44)); % Hay 15 datos cada dato se lee como un double

end

% Los datos recibidos son:

%

% referencia ref\_vel pwm velocidad vel\_enc ref\_pos posición pos\_gray error\_v int\_v der\_v error\_p int\_p der\_p binario

%

% r=datos(2);

u=datos(3); % Entrada PWM

y=datos(4); % Salida Velocidad

v=datos(5); % velocidad medida por el encoder

t=toc; % Tiempo transcurrido hasta ahora.

t\_new=t; % Tiempo en este instante.

% Escalón positivo: referencia +40, 4 segundos

if ((t\_new-t\_old)>t\_change && cambio==0) % Si se ha cumplido el tiempo hasta el nuevo cambio de referencia

r=40;

t\_change=4;

cambio=1;

% Envío de la nueva referencia:

fprintf(serialUSB,'ref,0,%f',r);

t\_old=t\_new; % Se actualiza el tiempo para el siguiente cambio de referencia.

fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\bTiempo transcurrido: %3.0f de %3.0f segundos',t,t\_ensayo);

end

% Escalón negativo: referencia -40, 4 segundos

if ((t\_new-t\_old)>t\_change && cambio==1) % Si se ha cumplido el tiempo hasta el nuevo cambio de referencia

r=-40;

t\_change=4;

cambio=2;

% Envío de la nueva referencia:

fprintf(serialUSB,'ref,0,%f',r);

t\_old=t\_new; % Se actualiza el tiempo para el siguiente cambio de referencia.

fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\bTiempo transcurrido: %3.0f de %3.0f segundos',t,t\_ensayo);

end

% Parada: referencia 0, 2 segundos

if ((t\_new-t\_old)>t\_change && cambio==2) % Si se ha cumplido el tiempo hasta el nuevo cambio de referencia

r=0;

t\_change=2;

cambio=0;

% Envío de la nueva referencia:

fprintf(serialUSB,'ref,0,%f',r);

t\_old=t\_new; % Se actualiza el tiempo para el siguiente cambio de referencia.

fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\bTiempo transcurrido: %3.0f de %3.0f segundos',t,t\_ensayo);

end

% Se guardan los datos de todas las variables en vectores

R(k) = r;

U(k) = u;

Y(k) = y;

V(k) = v;

T(k) = t;

end

fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\bTiempo transcurrido: %3.0f de %3.0f segundos',t,t\_ensayo);

% ENSAYO es una matriz que contendrá todos los datos del ensayo en cinco

% columnas como las del fichero en el que se guardan los datos del ensayo.

ENSAYO=[T;R;U;Y;V]'; % Resultados del ensayo

% PUESTA A CERO FINAL

%====================

r=0; % Referencia a cero

fprintf(serialUSB,'ref,0,%f',r);

fprintf(serialUSB,'rst,0,0'); % Comando 'rst': el ARDUINO se pone en los valores por defecto.

fclose(serialUSB);

delete(serialUSB);

clear serialUSB; % Libera el handler 'serialUSB' del puerto serie USB

% GRÁFICA DEL ENSAYO

%===================

figure(1);

subplot(2,1,1);

plot(T,[U;Y]); % PWM/Velocidad(rpm) (o plot(T,[R;U;Y]); Referencia/PWM/Velocidad)

title('Resultado del experimento')

legend('Entrada V', 'Salida RPM')

% FICHERO DE DATOS DEL ENSAYO:

%=============================

% Es fichero de texto de cinco columnas cuyo nombre son la fecha y hora

% del ensayo:

%

% Columna 1 Columna 2 Columna 3 Columna 4 Columna 5

%

% Tiempo Referencia PWM Velocidad Vel\_encoder

% (segundos) (rpm) (+-2.5V) (rpm) (rpm)

nombre=['tunningFB\_',datestr(now,30),'.txt']; % El nombre es la fecha y hora actual

f\_out=fopen(nombre, 'w'); % Abre para escritura 'w'

for i=1:length(ENSAYO) % Guardar datos en el fichero

fprintf(f\_out, '%f %f %f %f %f \n' ,ENSAYO(i,:));

end;

fclose(f\_out); % Cerrar el fichero de escritura

% IDENTIFICACIÓN

%===============

% Los datos se obtienen a intervalos de aproximadamente 30ms (dependerá de

% múltiples factores) pero no están muestreados de manera uniforme. Se

% remuestrean con un tiempo de muestreo Tm fijo.

ds=timeseries(ENSAYO(:,3:4),ENSAYO(:,1)); % Objeto timeseries para remuestrear los datos

Tm=0.01; % Tiempo de muestreo elegido en segundos

tiempo=[0:Tm:11.80]; % Nuevo vector de tiempo de 0 a 11.8 segundos

ds=resample(ds,tiempo); % Remuestrea los datos a un Tm uniforme

datos\_id=iddata(ds.Data(:,2),ds.Data(:,1),Tm); % iddata( Salida Velocidad(rpm) , Entrada PWM , Tiempo de muestreo)

sys1=idproc('P1'); % Estructura para identrificar un sistema de primer orden

sys2=idproc('P2'); % Estructura para identrificar un sistema de segundo orden

Gest2=pem(datos\_id,sys2); % Identificación del sistema de segundo orden

Gest1=pem(datos\_id,sys1); % Identificación del sistema de primer orden

% RESULTADOS POR PANTALLA

%========================

s=tf('s');

fprintf('\n \nFunción de Transferencia de segundo orden:')

G2=Gest2.K.value/(Gest2.Tp1.value\*s+1)/(Gest2.Tp2.value\*s+1) % FdT segundo orden

[pole,zero]=pzmap(G2);

fprintf('K=%f p1=%f p2=%f ',Gest2.K.value,pole(2),pole(1))

fprintf('\n \nFunción de Transferencia de primer orden:')

G1=Gest1.K.value/(Gest1.Tp1.value\*s+1) % FdT primer orden

K=Gest1.K.value ; % Ganancia K del sistema de primer orden

T=Gest1.Tp1.value ; % Constante de tiempo T del sistema de primer orden

fprintf('K=%f T=%f p1=%f \n',K,T,-1/T)

% COMPARACION G1 G2 A LA RESPUESTA ESCALÓN

%=========================================

subplot(2,1,2);

step(G1,G2);

title('Respuesta al escalón de G1 y G2')

legend('G1 1er. Orden', 'G2 2º Orden')

% REGULADORES

%=========================================

Rvp=(20-1/T)\*T/K; % Regulador de velocidad proporcional

Mvp=feedback(Rvp\*G1,1);

fprintf('\n \nRegulador de velocidad P: K=%f \n',Rvp)

fprintf('\nRegulador de velocidad PI:')

Rvpi=pidstd(20\*T/K,T) % Regulador de velocidad proporcional

Mvpi=minreal(feedback(Rvpi\*G1,1));

%fprintf('\nKp=%f Ti=%f Ki=%f \n',20\*T/K,T,1/T)

figure(2);

subplot(2,1,1);

step(Mvp,Mvpi);

title('Control de velocidad')

legend('Regulador P', 'Regulador PI')

Rp=1/(24\*T\*K); % Regulador de posición proporcional

Mp=minreal(feedback(Rp\*G1\*6/s,1));

fprintf('\n \nRegulador de posición P (un lazo): K=%f \n',Rp)

Rpc=100\*T/(6\*K\*Rvp); % Regulador de posición en cascada

Mpc=minreal(feedback(Rpc\*Mvpi\*6/s,1));

fprintf('\n \nRegulador de posición P (cascada): K=%f \n',Rpc)

subplot(2,1,2);

step(Mp,Mpc);

title('Control de posición')

legend('Regulación con un lazo', 'Regulación en cascada')

## medidas

% Script "medidas"

%

% Realiza las medidas de un módulo de FeedBack mediante la conexión

% del ARDUINO del módulo a un puerto USB del ordenador.

%

% Proporciona los rangos de tension de los potenciómetros de referencia

% y posicion y de la salida de velocidad de la tacodinamo. También se

% estiman los valores de tensión mínima (negativa) y máxima (positiva) de

% los bornes de la tacodinámo.

%

% REF POS VEL TACO RPM

% Max 4.833889 5.000000 3.577689 3.990756 48.123456

% Min 0.246111 0.000000 1.608022 -3.887911 -48.123456

%

% Es necesario mover manualmente el potenciometro del eje de entrada

% (referencia) para que se pueda medir su rango de tensiones. Primero el

% motor se mueve 5 segundos CW y 5 segundos CCW y a continuación se dispone

% de 10 segundos para mover el eje de entrada manualmente 2 o 3 veces del

% tope de un lado al otro.

clear referencia;

clear posicion;

clear velocidad;

clear vel\_encoder;

r=0; % referencia (rpm)

k=0; % índice de los vectores de datos

% SETUP

clear serialUSB; % Libera el handler 's' del puerto serie si estaba usado

newobjs = instrfind; % Obtiene los objetos serie activos

if length(newobjs)~=0 fclose(newobjs); end % Cierra los objetos serie que hayan quedado abiertos

% % Instrucciones que solo funcionan en versiones modernas de MATLAB R2020

% listaUSB=serialportlist; % Lista de dispositivos serie conectados 'a los puertos USB'

% serialUSB = serialport(listaUSB(1),115200); % Abre el puerto serie (1): 'COM3', 'COM5', etc.

%(habitualmente el USB del ARDUINO, si no hay más dispositivos conectados por puerto serie)

% Para ver los puertos, desde una ventana de comandos de Windows

% se puede ejecutar C:\> mode ( >>!mode )

% Usando serial(), fopen(), fprintf(), etc. aparentemente la comunicación es

% más rápida y el programa compatible con versiones de MATLAB anteriores a

% R2020, aunque serial() NO SE RECOMIENDA desde R2019.

% Por lo tanto se sustituyen los comandos comentados en las líneas

% anteriores por:

% Para ver los puertos en uso, desde una ventana de comandos de Windows

% se puede ejecutar C:\> mode ( >>!mode )

[status,cmdout] = system('mode'); % cmdout tiene la salida por pantalla del comando

pos=strfind(cmdout,'COM'); % Posición de los 'COM' en el texto, COM1 y el COM del ARDUINO

port=cmdout(pos(length(pos)):pos(length(pos))+3); % Se extrae la cadena del segundo (último) 'COM', habitualmente el USB

% port=cmdout(pos(length(pos)):pos(length(pos))+4); % +4 si el puerto es COM10 o superior (C+4 caracteres)

% del ARDUINO en los PCs del laboratorio, si no hay más dispositivos conectados

% por puerto serie y teniendo en cuenta que los PCs del laboratorio tienen un puerto serie

% COM1 no usado.

serialUSB = serial(port,'BaudRate',115200, 'Terminator',44);

% serialUSB = serial('COM12','BaudRate',115200, 'Terminator',44);

% Terminador ',' (ascii 44) para que al leer las líneas se reconozca cada dato que envía el ARDUINO

serialUSB.timeout=1;

fopen(serialUSB);

pause(1); % Necesario para que el puerto USB del Arduino MEGA (chino) esté listo

% % Instrucciones que solo funcionan en versiones modernas de MATLAB R2020

% configureTerminator(serialUSB,0); % Terminador que se envía tras cada writeline() para que el ARDUINO

% writeline(serialUSB,'rst,0,0'); % Comando 'rst': el ARDUINO se pone en los valores por defecto.

% writeline(serialUSB,'con,1,0'); % Comando 'con': el ARDUINO se pone en modo control de velocidad.

% writeline(serialUSB,'ppc,1,0'); % Comando 'ppc': el ARDUINO acepta los valores de referencia de velocidad desde el PC

% % La referencia de velocidad será en rpm

% % estimadas en lazo abierto.

% str\_ref=compose("ref,0,%f",100); % Comando "ref": Indica el valor de referencia en rpm.

% writeline(serialUSB,str\_ref); % Se le da un valor de 100 o -100 por encima del rango

% para asegurar el máximo.

% writeline(serialUSB,"sim,1,0"); %Comando 'sim': el ARDUINO se pone en modo simulación,para pruebas

fprintf(serialUSB,'rst,0,0'); % Comando 'rst': el ARDUINO se pone en los valores por defecto.

% fprintf(serialUSB,'sim,1,0'); %Comando 'sim': el ARDUINO se pone en modo simulación,para pruebas

fprintf(serialUSB,'ppc,1,0'); % Comando 'ppc': el ARDUINO acepta los valores de referencia de velocidad desde el PC

fprintf(serialUSB,'con,1,0'); % Comando 'con': el ARDUINO pone el control de velocidad en lazo abierto

% La referencia de velocidad será en rpm

% estimadas en lazo abierto.

fprintf(serialUSB,'ref,0,%f',100); % Comando 'ref': Indica el valor de referencia en rpm.

% Se le da un valor de 100 o -100 por encima del rango

% para asegurar el máximo.

% Medidas de potenciometros y taco.

tic; % inicio de medida de tiempo trancurrido.

disp('Espere: 5 segundos a máxima velocidad CW - ANOTE RPM MÁXIMA DEL DISPLAY')

while toc<5 % 5 segundos a velocidad CW máxima

k=k+1;

pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

% writeline(serialUSB,"dat,0,0"); % Comando "dat" para que el ARDUINO devuelva sus datos

fprintf(serialUSB,'dat,0,0'); % Comando 'dat' para que el ARDUINO devuelva sus datos

% configureTerminator(serialUSB,44); % Terminador "," (ascii 44) para que al leer las líneas se reconozca cada dato que envía el ARDUINO

for i=1:15

% datos(i) = double(readline(serialUSB)); % Hay 15 datos cada dato se lee como un double

datos(i) = str2double(fscanf(serialUSB,'%s',44)); % Hay 15 datos cada dato se lee como un double

end

%configureTerminator(serialUSB,0); % Volver al terminador estándar

% Los datos recibidos son:

%

% referencia ref\_vel pwm velocidad vel\_enc ref\_pos posición pos\_gray error\_v int\_v der\_v error\_p int\_p der\_p binario

%

velocidad(k)=datos(4);

vel\_encoder(k)=datos(5);

posicion(k)=datos(7);

end

% str\_ref=compose("ref,0,%f",-100); % Comando "ref": Indica el valor de referencia en rpm.

% writeline(serialUSB,str\_ref);

fprintf(serialUSB,'ref,0,%f',-100); % Comando 'ref': Indica el valor de referencia en rpm.

% Se le da un valor de 100 o -100 por encima del rango

% para asegurar el máximo.

disp('Espere: 5 segundos a máxima velocidad CCW - ANOTE RPM MÍNIMA DEL DISPLAY')

while toc<10 % 5 segundos a velocidad CW máxima

k=k+1;

pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

%writeline(serialUSB,"dat,0,0"); % Comando "dat" para que el ARDUINO devuelva sus datos

%configureTerminator(serialUSB,44); % Terminador "," (ascii 44) para que al leer las líneas se reconozca cada dato que envía el ARDUINO

fprintf(serialUSB,'dat,0,0');

for i=1:15

% datos(i) = double(readline(serialUSB)); % Hay 15 datos cada dato se lee como un double

datos(i) = str2double(fscanf(serialUSB,'%s',44)); % Hay 15 datos cada dato se lee como un double

end

%configureTerminator(serialUSB,0); % Volver al terminador estándar

velocidad(k)=datos(4);

vel\_encoder(k)=datos(5);

posicion(k)=datos(7);

end

% configureTerminator(serialUSB,0); % Terminador que se envía tras cada writeline() para que el ARDUINO

% writeline(serialUSB,"ppc,0,0"); % Comando "ppc": el ARDUINO acepta los valores de referencia de velocidad del potenciómetro

fprintf(serialUSB,'ppc,0,0'); % Comando 'ppc': el ARDUINO acepta los valores de referencia de velocidad desde el potenciómetro

l=0;

disp('ATENCIÓN: gire el eje de entrada manualmente 2 ó 3 veces hasta sus topes a izquierda y derecha')

while toc<11 % Delay de 1 segundo para dar tiempo a cambiar a referencia del potenciómetro.

end

while toc<20 % 9 segundos para mover el potenciómetro de referencia a sus topes.

k=k+1;

l=l+1;

pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

%writeline(serialUSB,"dat,0,0"); % Comando "dat" para que el ARDUINO devuelva sus datos

%configureTerminator(serialUSB,44); % Terminador "," (ascii 44) para que al leer las líneas se reconozca cada dato que envía el ARDUINO

fprintf(serialUSB,'dat,0,0'); % Comando 'dat' para que el ARDUINO devuelva sus datos

for i=1:15

% datos(i) = double(readline(serialUSB)); % Hay 15 datos cada dato se lee como un double

datos(i) = str2double(fscanf(serialUSB,'%s',44)); % Hay 15 datos cada dato se lee como un double

end

%configureTerminator(serialUSB,0); % Volver al terminador estándar

referencia(l)=datos(1);

velocidad(k)=datos(4);

vel\_encoder(k)=datos(5);

posicion(k)=datos(7);

end

% Puesta a cero final

% configureTerminator(serialUSB,0);

% str\_ref=compose("ref,0,%f",r); % Comando 'ref': Indica el valor de referencia en rpm.

% writeline(serialUSB,str\_ref);

% writeline(serialUSB,'rst'); % Comando 'rst': el ARDUINO se pone en los valores por defecto.

fprintf(serialUSB,'ref,0,%f',0); % Comando 'ref': Indica el valor de referencia en rpm.

pause(0.010); % Pausa de "n.nnn" segundos (Arduino MEGA)

fprintf(serialUSB,'rst,0,0'); % Comando 'rst': el ARDUINO se pone en los valores por defecto.

fclose(serialUSB);

delete(serialUSB); % Libera el handler 's' del puerto serie

clear serialUSB; % Libera el handler 's' del puerto serie USB

% RESULTADOS

% Max\_REF=max(referencia)\*2.5/45+2.5;

% Min\_REF=min(referencia)\*2.5/45+2.5+0.07;

Max\_REF=max(referencia);

Min\_REF=min(referencia);

Max\_POS=max(posicion)\*2.5/180+2.5;

Min\_POS=min(posicion)\*2.5/180+2.5;

Max\_VEL=max(velocidad)\*0.95/45+2.55;

Min\_VEL=min(velocidad)\*0.95/45+2.55;

Max\_TACO=(Max\_VEL-0.08)\*20/5-10;

Min\_TACO=(Min\_VEL-0.08)\*20/5-10;

Max\_RPM=max(vel\_encoder);

Min\_RPM=min(vel\_encoder);

disp(' REF POS VEL TACO RPM');

fprintf('Max %f %f %f %f %f \n', Max\_REF, Max\_POS, Max\_VEL, Max\_TACO, Max\_RPM);

fprintf('Min %f %f %f %f %f \n', Min\_REF, Min\_POS, Min\_VEL, Min\_TACO, Min\_RPM);

## comandoFB

function comandoFB(comando,numero1,numero2)

% función "comandoFB"

% Envía el comando al ARDUINO y devuelve por pantalla la respuesta si procede.

%

% J.A. Sirgo 18/03/2021

%

% >> comandoFB('cmd',int,float)

% cmd: es el comando que se quiere envial al ARDUINO.

% int: valor entero

% float: valor del "float" que requieren algunos comandos.

% (poner 0 en int y float si no se requieren)

%

% Comandos aceptados:

%

% Comandos sin parámetros:

% dat - Petición de datos desde el PC. Se recibirán datos de distintas variables.

% rst - Se ponen los parámetros por defecto. Petición del PC al iniciar o cerrar la conexión

% est - Petición del estado de los modos de control, referencia, realimentacion, etc.

%

% Comandos con parámetro entero binario (0, 1):

% mod - Selecciona el modelo de Feedback original=0, modificado=1

% sim - Funcionamiento en modo normal=0, simulación=1

% ppc - Selecciona la señal de referencia desde el potenciómetro de la unidad Feedback=0 o desde el PC=1

% fbv - Realimentación de velocidad por tacodinamo=0 o encoder=1

% fbp - Realimentación de posición por potenciometro=0 o disco Gray=1

%

% Comandos con parámetro entero:

% con - Modo de control: vel lazo abierto=1, vel lazo cerrado=2, pos un lazo simple=3, pos en cascada=4

%

% Comandos con parámetros entero y real:

% ref - Valor de la referencia en el valor "float" desde el PC: velocidad=0 o posición=1

% pid - Valores de las constantes de los reguladores PID en el valor "float":

% 1=Kp de regulador de velocidad

% 2=Ki de regulador de velocidad

% 3=Kd de regulador de velocidad

% 4=Kp de regulador de posición

% 5=Ki de regulador de posición

% 6=Kd de regulador de posición

% sin - Generador de referencias senoidales. Valor del "int":

% 0=desactivar

% 1=activar

% 2=valor de W en hercios (el valor "float" indica la W frecuencia en hercios)

% 3=valor de A en voltios (el valor "float" indica la A amplitud en voltios)

%

% SETUP:

%=======

% Configuración del puerto serie USB donde está conectado el ARDUINO

clear serialUSB; % Libera el handler 'serialUSB' del puerto serie si estaba usado

newobjs = instrfind; % Obtiene los objetos serie activos

if length(newobjs)~=0 fclose(newobjs); end % Cierra los objetos serie que hayan quedado abiertos

% Para ver los puertos en uso, desde una ventana de comandos de Windows

% se puede ejecutar C:\> mode ( >>!mode )

[status,cmdout] = system('mode'); % cmdout tiene la salida por pantalla del comando

pos=strfind(cmdout,'COM'); % Posición de los 'COM' en el texto, COM1 y el COM del ARDUINO

port=cmdout(pos(length(pos)):pos(length(pos))+3); % Se extrae la cadena del segundo (último) 'COM', habitualmente el USB

% del ARDUINO en los PCs del laboratorio, si no hay más dispositivos conectados

% por puerto serie y teniendo en cuenta que los PCs del laboratorio tienen un puerto serie

% COM1 no usado.

serialUSB = serial(port,'BaudRate',115200, 'Terminator',44); % Se configura el puesto serie.

serialUSB.timeout=1;

% serialUSB.Terminator=44; % Terminador ',' (ascii 44) para que al leer las líneas se reconozca cada dato que envía el ARDUINO

fopen(serialUSB); % Abre el puerto serie: 'COM3', 'COM4', 'COM5', etc.

pause(1); % Necesario para que el puerto USB del Arduino MEGA (chino) esté listo

% ENVIO DEL COMANDO:

% ====================

mensaje=sprintf('%s,%d,%f',comando,numero1,numero2);

fprintf(serialUSB,mensaje); % Comando para el ARDUINO

% RECEPCION DE RESPUESTAS A COMANDOS ‘dat’ Y ‘est’

%=================================================

% Si el comando es 'dat'

if comando(1)=='d'

for i=1:15

datos(i) = str2double(fscanf(serialUSB,'%s',44)); % Hay 15 datos cada dato se lee como texto y se pasa a un double

end

disp('referencia ref\_vel pwm velocidad vel\_enc ref\_pos posición pos\_gray error\_v int\_v der\_v error\_p int\_p der\_p binario')

disp(datos); % Mostrar los datos

end

% Si el comando es 'est'

if comando(1)=='e'

for i=1:6 % hay 6 datos

estado = fscanf(serialUSB); % Cada dato de estado se lee como un string

disp(estado); % Mostrar el estado

end

end

fclose(serialUSB);

delete(serialUSB); % Libera el handler 'seriealUSB' del puerto serie

clear serialUSB;