Capítulo 5: Desarrollo del programa informático

5.1. Introducción al Data Adquisition de Matlab.

El motor de adquisición de datos (o simplemente motor) es un *MEX-file* o enlace de librería dinámica *"dynamic link library"* (DLL) que:

- Almacena los objetos de los dispositivos y les asocia valores a las propiedades que controlan la aplicación de adquisición de datos.
- Controla la sincronización de eventos.
- Controla el almacenamiento de los datos adquiridos o puestos en cola.

Mientras el motor realiza estas tareas, puedes usar *Matlab* para otras labores tales como el análisis de los datos adquiridos. En otras palabras, el motor y *Matlab* son asíncronos.

5.1.1. Adquiriendo datos analógicos.

Si tienes una tarjeta de sonido instalada, puedes ejecutar el siguiente ejemplo:

```
%Crea el objeto del dispositivo--Crea el objeto entrada analogica para
la tarjeta de sonido
ai = analoginput('winsound');
%Añadir canales--Añade dos canales de hardware en ai
addchannel(ai,1:2);
%Valores de configuración de las propiedades--Configura la tasa de
muestreo a 44,1 kHz y recoge 1 segundo de datos (44100 muestras) por
cada canal
set(ai,'SampleRate',44100)
set(ai,'SamplesPerTrigger',44100)
%Adquirir datos--Comienza la adquisición. Cuando todos los datos se
han adqurido, ai automaticamente se detiene.
start(ai)
data = getdata(ai);
plot(data)
%Limpiar memoria--Cuando ya no necesites ai, se debe borrar de la
memoria y del Matlab workspace.
delete(ai)
clear ai
```

5.1.2. Sacando datos analógicos.

Si tienes una tarjeta de sonido instalada, puedes ejecutar el siguiente ejemplo:

```
%Creando el objeto del dispositivo--Crea el objeto salida analogica ao
para la tarjeta de sonido
ao = analogoutput('winsound');
%Añadir canales--Añade dos canales de hardware a ao
addchannel(ao,1:2);
```


```
%Configurar los valores de las propiedades--Configura la tasa de
muestreo a 44,1 kHz para cada canal
set(ao,'SampleRate',44100)
%Sacando datos--Crea 1 segundo de salida de datos, y almacena los
datos en el motor para una eventual salida hacia el subsistema de
salida analogica.
%Debes almacenar una columna de datos por cada canal de hardware
añadido.
data = sin(linspace(0,2*pi*500,44100)');
putdata(ao,[data data])
%Comienza la salida. Cuando todos los datos han salido, ao
automáticamente deja de  ejecutarse
start(ao)
%Limpiar--Cuando ya no necesites ao, debes borrarla de la memoria y
del Matlab workspace
delete(ao)
clear ao
```

5.1.3. Leyendo y escribiendo valores digitales.

Si tienes una tarjeta *National Instrument* compatible puedes ejecutar el siguiente ejemplo, que escribe y lee valores digitales.

```
%Crea el objeto del dispositivo--Crea el objeto digital I/O dio para
una
%tarjeta National Instrument con hardware ID 1
dio = digitalio('nidaq',1);
%Añade lineas--Añade cuatro lineas de hardware y las configura como
lineas
%de salida y dos como entrada
addline(dio,0:3,0,'out')
addline(dio,4:5,1,'in')
%0:3--Lineas de la cero a la tres
%4:5--Lineas cuatro y cinco
%0--puede valer cero o uno, depende del dispositivo y sirve para
indicar si
%es una linea de entrada o de salida. Si no se corresponde con 'out' o
'in'
%da un mensaje de error

%Escribimos valores usando el comando PUTVALUE
putvalue(dio,[0 1 0 0]);
%Leemos valores usando el comando GETVALUE
getvalue(dio);
%Limpiamos--Cuando no necesitemos dio, lo borramos de la memoria y del
%Matlab workspace
```

5.1.4. Trabajando con datos

Estas son las funciones disponibles en el *Data Adquisition* con las que adquirir datos mediante nuestro dispositivo.

función	Finalidad	AI	AO	DIO
flushdata	Borra los datos del motor de adquisición de datos	√		
getdata	Extrae datos, tiempo e información del evento desde el motor de adquisición de datos	√		
getsample	Adquiere inmediatamente una muestra	√		
getvalue	Lee valores de las líneas			√
peekdata	Muestra el dato adquirido mas recientemente	√		
putdata	Pone datos a la cola del motor para una eventual salida		√	
putsample	Saca una muestra inmediatamente		√	
putvalue	Escribe valores a las líneas			√

Tabla 5. 1. Funciones pertenecientes al Data Adquisition dedicadas al manejo de datos mediante cualquier dispositivo de adquisición.

Las funciones que más se adecúan al trabajo que deseamos realizar son:

Getsample:

Para adquirir muestras analógicas ya que no bloquea la ejecución del programa hasta adquirir las muestra como hace getdata pero se asegura de adquirir las muestras deseadas e ir mostrándolas sin perder o repetir alguna, como ocurre con la función peekdata.

Putsample:

Es similar a getsample pero para la salida de muestras analógicas.

Putvalue:

Es la encargada de enviar salidas digitales a la tarjeta de datos [14].

5.2. Adquisición de datos analógicos (Comprobando conexión de los sensores).

Este pequeño *Script* nos será útil cuando conectemos algún nuevo sensor a las entradas analógicas del dispositivo de adquisición de datos, con él podemos realizar una prueba y ver si los resultados se corresponden con lo que debe marcar el sensor, si no es así, posiblemente, la conexión del sensor no es la correcta y puede que esté apareciendo una *señal de ruido* junto a la señal del sensor.

Debemos indicar el canal en el cual deseamos tomar medidas, en este ejemplo aparece indicado el canal 0. Si el dispositivo utilizado no es de *National Instrument* debemos indicar el driver de la tarjeta y el número de Id puede variar.

```
%            ('driver',nº de ID)
ai = analoginput('nidaq',1)
%            (objeto, nº de canal)
chans_i = addchannel(ai,0)

set(ai,'SampleRate',30)
set(ai,'SamplesPerTrigger',300)

set(chans_i,'SensorRange',[0 10])
set(chans_i,'InputRange',[0 10])
set(chans_i,'UnitsRange',[0 10])

start(ai);

h=getdata(ai,300);
plot(h);

stop(ai);
delete(ai);
```

En la ilustración 5.1. vemos una señal de 10V proveniente de un sensor, totalmente distorsionada por el ruido, debido a que la conexión se debería haber efectuado en modo diferencial en lugar de RSE (ver apartado 4.2.).

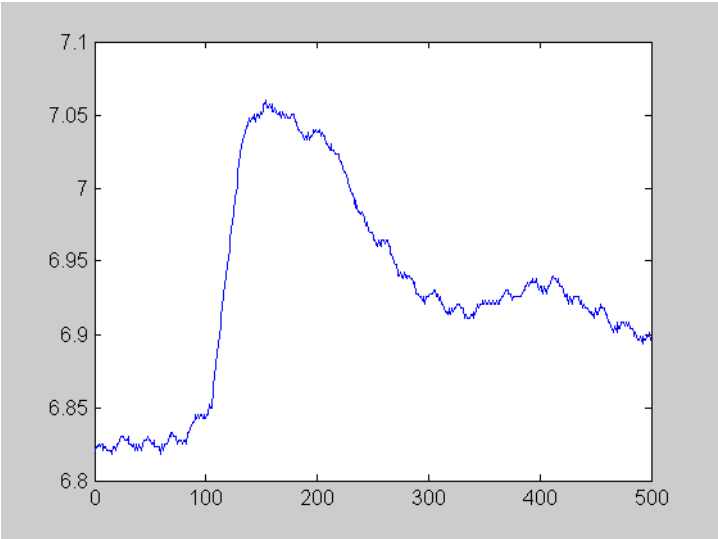


Ilustración 5. 1. Señal de 10V con ruido (entrada simple en lugar de diferencial).

5.3. Programación de los controladores.

5.3.1. Introducción.

En este y los subapartados que vienen a continuación, dentro de la programación de los controladores, sólo veremos las funciones y el código necesario para el funcionamiento de distintos cilindros que componen el robot, pero no se

estudiarán los distintos controladores, esta tarea quedará aclarada más adelante en el capítulo 6 de la memoria.

5.3.2. Programación del controlador para la posición del cilindro DNC (Controlador todo-nada).

A continuación se muestra el código necesario para controlar la posición de un cilindro, en concreto el cilindro *DNC-32-100-PPV-A-S2* al que le hemos acoplado un sensor externo *SMAT-8E-S50-IU-M8* para conocer su posición y que se desplaza mediante el accionamiento de dos válvulas *MOFH-3-1/8* encargada cada una de moverle en un sentido.

El código necesario está formado por tres funciones *PID, callback_PID* y *finaliza* que se hallan recogidas dentro de la carpeta *posiciónDNC* en el CD-ROM del proyecto. Estos nombres son genéricos e iguales para todos los controladores utilizados; sin embargo, el controlador en este caso no es *PID* sino *on-off.*

Para ejecutar la aplicación primero debemos llamar a la función PID escribiendo [ai,dio] = PID()

en el *Command Window* de *Matlab* con lo que se ejecutará la función:

```
function [ai,dio] = PID()
clear all

%declara las variables
global S_REFERENCIA
global REFERENCIA
global SALIDA
global N_MUESTRA
global S_CONTROL

%Asignación dinámica de las variables
S_CONTROL=[];
S_REFERENCIA=[];
SALIDA=[];
REFERENCIA=[];

%inicia N_MUESTRA con valor uno
N_MUESTRA = 1;

%crea los objetos dio y ai y asigna los canales
dio = digitalio('nidaq',1);
ai = analoginput('nidaq',1);
chans_o = addline(dio,0:1,0,'out');
chans_i = addchannel(ai,0);
```


```
%Ajusta los rangos de las entradas a los de los sensores
set(chans_i,'SensorRange',[0 10]);
set(chans_i,'InputRange',[0 10]);
set(chans_i,'UnitsRange',[0 10]);


% Frecuencia de muestreo (Hertzios)
set(ai,'SampleRate',30);
ai.SamplesAcquiredFcnCount = 1;
% Función que implementa el algoritmo de control
ai.SamplesAcquiredFcn = {@callback_PID, dio};
%Numero de muestras que se van a adquirir en cada disparo "Trigger"
set(ai,'SamplesPerTrigger',300);
%set(ai,'TriggerRepeat',inf);
start(ai);

tic
```

Como podemos ver desde la función *PID* se llama continuamente a la función *callback_PID* por cada muestra adquirida, aquí vemos la función callback:

```
function callback_PID(obj,event, dio)
%Declaración de variables
global S_REFERENCIA
global REFERENCIA
global SALIDA
global N_MUESTRA
global S_CONTROL
persistent sal

%Asignación de valores a las variables
REFERENCIA(N_MUESTRA)=6;
Kp = 1;
sal=getsample(obj);
SALIDA(N_MUESTRA) = sal;
S_REFERENCIA(N_MUESTRA)=REFERENCIA(N_MUESTRA);

%Posición
Y_k = sal;
%Error
E_k = REFERENCIA(N_MUESTRA)-Y_k;

%Señal de control
U = Kp * E_k;

    %Para -0.5<U<0.5 consideramos el error aceptable
    if U>0.5
        A=0;
        B=1;

    elseif U<-0.5
        A=1;
```

```
        B=0;

    else
        A=1;
        B=1;
    end

S_CONTROL(1:2,N_MUESTRA)= [A B];
putvalue(dio,[A B]);
N_MUESTRA=N_MUESTRA + 1;
```

Como las válvulas que estamos usando son de accionamiento *todo-nada*, el control no posee término integral ni derivativo, sino que sólo existe la posibilidad de controlar mediante una constante proporcional.

En cualquier momento podemos detener la ejecución del programa llamando a la función *finaliza* que elimina el objeto *ai* y *dio* y muestra la salida y la señal de referencia por pantalla, para ello escribimos *finaliza(ai,dio)* en la ventana de comandos de *Matlab*:

```
function finaliza(ai, dio)

%Declaracion de variables
global SALIDA;
global S_REFERENCIA;

%Devuelve el cilindro a la posicion inicial
putvalue(dio,[1 0]);

%Para y elimina ai y dio
stop(ai);
stop(dio);
delete(ai);
delete(dio);

%Muestra los resultados por pantalla
figure
plot(SALIDA);
hold on;
plot(S_REFERENCIA,'r');
```

5.3.3. Programación de un controlador individual para el cilindro DNCI (Control analógico).

Los cilindros *DNCI-32-300-P-A* son accionados mediante las válvulas proporcionales *MPYE-5-1/8-LF-010-B* que, a diferencia del caso anterior, controlan

tanto el sentido de avance del cilindro como el caudal de aire que introducen en cada momento a los cilindros; aunque en el robot funcionarán los dos conjuntamente, dentro de la carpeta *posicionDNCI* se encuentran recogidas las funciones necesarias para el adecuado funcionamiento de uno solo de los cilindros *DNCI* esto es útil para hacer pruebas con un único cilindro (téngase en cuenta que el comportamiento de dos cilindros unidos mediante un eslabón será distinto al que tenían por separado).

Igual que en el caso anterior llamamos a la función *PID* de la siguiente forma:

[ai, ao] = PID()

Con lo que se ejecuta lo siguiente:

```
function [ai, ao] = PID()
clear all

%declara las variables
global S_REFERENCIA;
global REFERENCIA;
global SALIDA;
global INTEGRAL;
global N_MUESTRA;
global S_CONTROL;

%Asignación dinámica de las variables
S_CONTROL=[];
S_REFERENCIA=[];
SALIDA=[];

%inicia N_MUESTRA e INTEGRAL
N_MUESTRA = 1;
INTEGRAL = 0;

%crea los objetos ai y ao y asigna los canales
ao = analogoutput('nidaq',1);
ai = analoginput('nidaq',1);
chans_o = addchannel(ao,0);
chans_i = addchannel(ai,0);

%Ajusta los rangos de las entradas a los de los sensores
set(chans_i,'SensorRange',[0 10]);
set(chans_i,'InputRange',[0 10]);
set(chans_i,'UnitsRange',[0 10]);

%ajusta los rangos de salida a los de los actuadores
set(chans_o,'OutputRange',[0 10]);
set(chans_o,'UnitsRange',[0 10]);


% Frecuencia de muestreo (Hertzios)
```

```
set(ai,'SampleRate',30);
ai.SamplesAcquiredFcnCount = 1;
% Función que implementa el algoritmo de control
ai.SamplesAcquiredFcn = {@callback_PID, ao};
%Numero de muestras que se van a adquirir en cada disparo "Trigger"
set(ai,'SamplesPerTrigger',inf);
%set(ai,'triggerrepeat',inf);

%putsample(ao, 0);
%start(ao);
start(ai);
tic
```

Lo cual es muy similar a la función *PID* anterior salvo que con el objeto de salida analógica *ao* en lugar de la salida digital *dio*, esto además nos obliga a indicar los rangos de salida deseados al igual que para *ai* ya que por defecto para este dispositivo de *National Instrument* el rango es [-5,5].

Vemos ahora la función *callback_PID* que es llamada continuamente para cada muestra de *ai* adquirida:

```
function callback_PID(obj,event, ao)

%Declaración de variables
global S_REFERENCIA;
global REFERENCIA;
global SALIDA;
global N_MUESTRA;
global INTEGRAL;
global S_CONTROL;
global U;
global sal;
persistent E_k1;

%E_K1 es el error en la muestra anterior para N_MUESTRA==1 valdra 0
if (N_MUESTRA == 1)
    E_k1 = 0;
end

%Situa el valor de la señal adquirida ai en la variable sal
sal=getsample(obj);

REFERENCIA(N_MUESTRA)=5;
%REFERENCIA=[3.5*ones(1,500) 7*ones(1,500)];

%valores del PID, podemos ajustarlo para obtener una respuesta
diferente
Kp = 1;
Ki = 0;
Kd = 0;
```


```
    SALIDA(N_MUESTRA) = sal;
    S_REFERENCIA(N_MUESTRA)=REFERENCIA(N_MUESTRA);

    %Posicion
    Y_k = sal;
    %Error
    E_k = REFERENCIA(N_MUESTRA)-Y_k;
    %Integral
    INTEGRAL = INTEGRAL + E_k;

    %Señal de control
    U = Kp * E_k  + Ki * INTEGRAL + Kd * (E_k - E_k1)/(1/30)

    %La señal de control oscila en torno al cero
    %Saturacion que mantiene la señal de control en valores entre -5 y
5
    if U>5
    U=5;
    elseif U<-5
    U=-5;
    end

    %Sumamos 5 para que el valor que introducimos al actuador sea
entre 0 y 10
    U=U+5;

    %Si la señal esta saturada evitamos que INTEGRAL aumente su valor
    if U>=9.8 | U<=0.2
    INTEGRAL = INTEGRAL - E_k;
    end

    %Para valores mayores a 9.8 o menores que 0.2 la valvula vibra
    if U>9.8
    U=9.8;
    elseif U<0.2
    U=0.2;
    end

    S_CONTROL(1,N_MUESTRA)= [U];
    putsample(ao, U);
    E_k1 = E_k;

    N_MUESTRA = N_MUESTRA + 1;
```

En este caso el sistema de control es el correspondiente al modelo PID básico (Ilustración 5.2.)

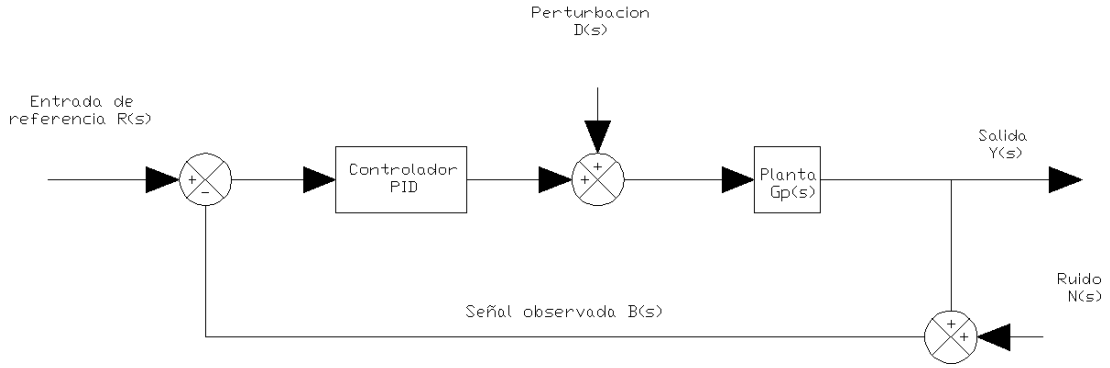


Ilustración 5. 2. Sistema controlado mediante PID.

En cualquier momento podemos detener la ejecución del programa llamando a la función *finaliza* que detiene y elimina el objeto *ai* y *ao* y muestra la salida, la señal de referencia y la señal de control por pantalla, para ello escribimos *finaliza(ai,ao)* en la ventana de comandos de *Matlab*:

```
function finaliza(ai, ao)
toc

%Declaracion de variables
global SALIDA;
global S_REFERENCIA;
global S_CONTROL;

%Devuelve el cilindro a la posición inicial
putsample(ao,4);

%Para y elimina los objetos ai y ao
stop(ai);
stop(ao);
delete(ai);
delete(ao);

%Muestra por pantalla los resultados
figure
plot(SALIDA);
hold on;
plot(S_REFERENCIA,'r');
hold on;
plot(S_CONTROL,'g');
```


5.3.4. Control simultáneo de los tres cilindros.

El código necesario para conseguir el funcionamiento de los tres cilindros simultáneamente está basado en los códigos que aparecen en los apartados *5.3.2 y 5.3.3* sólo que algunas variables aparecen por triplicado o en vectores con los que se podrán obtener los valores para cada uno de los cilindros mediante el acceso a la posición deseada dentro del vector. El código se encuentra dentro de la carpeta *controlsimultaneo* y está formado por la función *PID* que inicia la aplicación, la función *callback_PID* que contiene los controladores PID y la función *finaliza* que termina el programa además de tres funciones llamadas gráficas que muestran los resultados para cada uno de los cilindros. La ejecución del programa se realiza de la siguiente forma:

1º) Escribimos la llamada a la función PID en la ventana de comandos:

[ai, ao, dio] = PID()

```
function [ai, ao, dio] = PID()
clear all;

%Declaracion de variables
global REFERENCIA;
global N_MUESTRA;
global INTEGRAL;
global S_CONTROL1;
global S_CONTROL2;
global S_CONTROL3;
global S_REFERENCIA1;
global S_REFERENCIA2;
global S_REFERENCIA3;
global SALIDA1;
global SALIDA2;
global SALIDA3;

S_CONTROL1=[];
S_CONTROL2=[];
S_CONTROL3=[];
S_REFERENCIA1=[];
S_REFERENCIA2=[];
S_REFERENCIA1=[];
SALIDA1=[];
SALIDA2=[];
SALIDA3=[];

N_MUESTRA = 1;
INTEGRAL = [0 0 0];

%driver
```

```
%Advantech:advantech
%Data Translation:dtol
%National Instrument:nidaq
%asegurese que el numero de ID del dispositivo es el correcto
ai = analoginput('nidaq',1);
chans_i = addchannel(ai,[0:2]);
ao = analogoutput('nidaq',1);
chans_o = addchannel(ao,0:1);
dio = digitalio('nidaq',1);
chans_d = addline(dio,0:1,0,'out');

%Ajuste de los rangos
set(chans_i,'SensorRange',[0 10]);
set(chans_i,'InputRange',[0 10]);
set(chans_i,'UnitsRange',[0 10]);

set(chans_o,'OutputRange',[0 10]);
set(chans_o,'UnitsRange',[0 10]);

%Frecuancia de muestreo (en Hertzios)
set(ai,'SampleRate',30);
ai.SamplesAcquiredFcnCount = 1;
% Función que implementa el algoritmo de control
ai.SamplesAcquiredFcn = {@callback_PID, ao, dio};
set(ai,'SamplesPerTrigger',1000);
set(ai,'triggerrepeat',inf);

%putsample(ao,[0 0]);
%start(ao);
start(ai);

tic
```

Como vemos se realiza una llamada a la función *callback_PID* por cada muestra adquirida, abajo tenemos la función antes mencionada.

```
function callback_PID(obj,event, ao, dio)


global REFERENCIA;
global N_MUESTRA;
global INTEGRAL;
global S_CONTROL1;
global S_CONTROL2;
global S_CONTROL3;
global S_REFERENCIA1;
global S_REFERENCIA2;
global S_REFERENCIA3;
global SALIDA1;
global SALIDA2;
global SALIDA3;
persistent E_k1;
persistent sal;

sal = getsample(obj);
```


```
REFERENCIA([1;N_MUESTRA])=6;
REFERENCIA([2;N_MUESTRA])=5;
REFERENCIA([3;N_MUESTRA])=5;


    Kp(1)=1;
    Ki(1)=0;
    Kd(1)=0;

    Kp(2)=1;
    Ki(2)=0;
    Kd(2)=0;

    Kp(3)=1;
    Ki(3)=0;
    Kd(3)=0;

if (N_MUESTRA == 1)
    E_k1 = [0 0 0];
end

%empieza la implementación del cilindro 1

    SALIDA1(N_MUESTRA) = sal(3);
    S_REFERENCIA1(N_MUESTRA)=REFERENCIA(1);

    Y_k(1) = sal(3);
    E_k(1) = REFERENCIA(1)-Y_k(1);

    U(1) = Kp(1) * E_k(1);

    if U(1)>0.5
        A=0;
        B=1;

    elseif U(1)<-0.5
        A=1;
        B=0;
    else
        A=1;
        B=1;
    end


%ha finalizado la implementación del cilindro 1


%empieza la implementación del cilindro 2
    SALIDA2(N_MUESTRA) = sal(1)
    S_REFERENCIA2(N_MUESTRA)=REFERENCIA(2);


    Y_k(2) = sal(1);
```

```
    E_k(2) = REFERENCIA(2)-Y_k(2);
    INTEGRAL(2) = INTEGRAL(2) + E_k(2);

    U(2) = Kp(2) * E_k(2)  + Ki(2) * INTEGRAL(2) + Kd(2) * (E_k(2) -
E_k1(2))/(1/30);


    if U(2)>5
    U(2)=5;
    elseif U(2)<-5
    U(2)=-5;
    end


    U(2)=U(2)+5;


    if U(2)>=9.8 | U(2)<=0.2
    INTEGRAL(2) = INTEGRAL(2) - E_k(2);
    end

    if U(2)>9.8
    U(2)=9.8;
    elseif U(2)<0.2
    U(2)=0.2;
    end


%ha finalizado la implementación del cilindro 2


%empieza la implementación del cilindro 3

    SALIDA3(N_MUESTRA) = sal(2);
    S_REFERENCIA3(N_MUESTRA)=REFERENCIA(3);


    Y_k(3) = sal(2);
    E_k(3) = REFERENCIA(3)-Y_k(3);
    INTEGRAL(3) = INTEGRAL(3) + E_k(3);

    U(3) = Kp(3) * E_k(3)  + Ki(3) * INTEGRAL(3) + Kd(3) * (E_k(3) -
E_k1(3))/(1/30);

    if U(3)>5
    U(3)=5;
    elseif U(3)<-5
    U(3)=-5;
    end


    U(3)=U(3)+5;

    if U(3)>=9.8 | U(3)<=0.2
```


```
    INTEGRAL(3) = INTEGRAL(3) - E_k(3);
    end

    if U(3)>9.8
    U(3)=9.8;
    elseif U(3)<0.2
    U(3)=0.2;
    end
 %ha finalizado la implementación del cilindro 3

    S_CONTROL1(1:2,N_MUESTRA)= [A B];
    S_CONTROL2(1,N_MUESTRA)= [U(2)];
    S_CONTROL3(1,N_MUESTRA)= [U(3)];

   E_k1 = E_k;

    putvalue(dio,[A B]);
    putsample(ao, [U(2) U(3)]);

    N_MUESTRA=N_MUESTRA + 1;
```

El funcionamiento es el mismo que el de los apartados *5.3.2 y 5.3.3.* aunque funcionando tres controladores al mismo tiempo en lugar de solo uno.

2º) Cuando deseemos finalizar el programa escribimos:

finaliza(ai, ao, dio)

```
function finaliza(ai, ao, dio)
toc

putvalue(dio,[0 1]);
putsample(ao,[4,4]);

stop(ai);
stop(ao);
delete(ai);
delete(ao);
delete(dio);
```

En este caso la función finaliza no se encarga de mostrar los resultados ya que al existir 3 cilindros que presentar sería muy confuso representarlos los tres en una sola ventana o que la función desplegase una ventana para cada cilindro, por eso he optado por realizar tres funciones que muestran cada una los resultados de un cilindro concreto; así podremos representar en cada momento los resultados que deseemos.

3ª) Ahora podemos ver los resultados de cualquier cilindro mediante la llamada a las respectivas funciones encargadas de crear las gráficas:

graficas1(ai,ao)

graficas2(ai,ao)

graficas3(ai,ao)

Para el cilindro *DNC-32-100-PPV-A-S2:*

```
function graficas1(ai,ao)

global S_REFERENCIA1;
global SALIDA1;


figure()
plot(S_REFERENCIA1,'r');
hold on;

plot(SALIDA1);
hold on;
legend('Referencia','Posición');
xlabel('Numero de muestras')
ylabel('Voltaje, V')
```

Para uno de los cilindros *DNCI-32-300-P-A*

```
function graficas2(ai,ao)

global S_REFERENCIA2;
global SALIDA2;
global S_CONTROL2;

figure()
plot(S_REFERENCIA2,'r');
hold on;

plot(S_CONTROL2,'g');
hold on;

plot(SALIDA2);
hold on;
legend('Referencia','Señal de control','Posición')
xlabel('Numero de muestras')
ylabel('Voltaje, V')
```


El código de la función *graficas3* funciona igual al arriba mostrado *(graficas2)* sólo que trabaja con las variables terminadas en 3, correspondientes al otro cilindro *DNCI*, en lugar de 2.

5.4. Programa completo.

5.4.1. Creación y uso de la interfaz gráfica

El programa completo consta básicamente del programa anterior desarrollado para funcionar de manera interactiva con una interfaz gráfica de manejo sencillo e intuitivo, con lo que para indicar valores de PID, referencias o realizar gráficas con los resultados no haya que escribir líneas de código, sino sólo hacer unas pocas pulsaciones de ratón.

Matlab incluye una herramienta para la creación de gráficas de usuario denominada *GUIDE* (*Ilustración 5.3*), con la que he realizado los menús que forman el

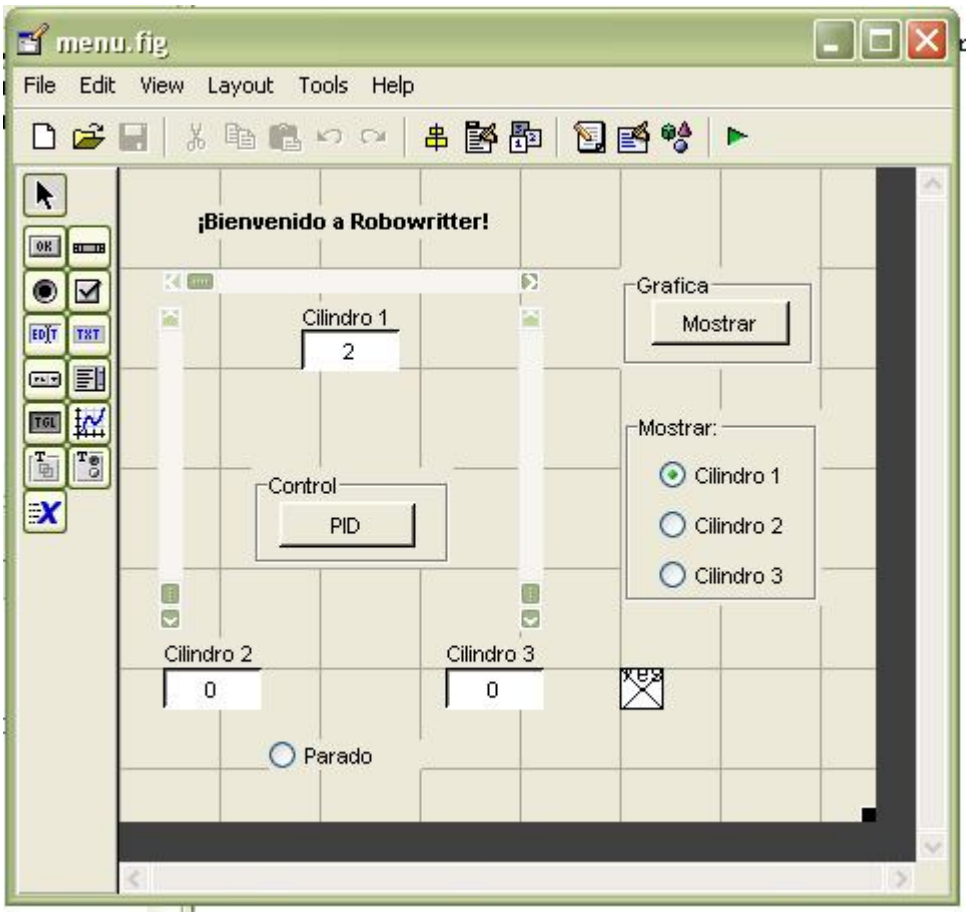


Ilustración 5. 3. Creación del "layout" mediante GUIDE.

programa.

La creación de cualquier menú gráfico se realiza mediante la utilización de las distintas herramientas y su colocación sobre el *"layout"* (capa) una vez colocados todos los elementos en el lugar deseado al grabar el menú nos aparece tanto un fichero *.fig* que es el *"layout"* como un *m-file* con las funciones creadas para cada tipo de botón incluido en el *"layout"*, ya no tenemos más que rellenar estas funciones con el código que deseemos que se ejecute en cada caso y grabar también este *m-file* con el mismo nombre que el fichero *.fig.*

En este caso primero he realizado una pantalla de presentación, con el nombre *Robowritter,* como vemos en la ilustración *5.4*, que es la que inicia todo el programa. Si ejecutamos esta función nos aparecerá lo siguiente:

Ilustración 5. 4. Pantalla de presentación del programa.

Cuando pulsamos con el ratón sobre el botón que pone "CONTINUAR" pasamos a la siguiente función que es una pantalla llamada "referencia" que nos pregunta si

deseamos establecer el punto de referencia de los dos cilindros *DNCI* y situar al cilindro *DNC* en posición retraída. Si es la primera vez que ejecutamos el programa desde que encendimos la fuente de alimentación, por lo general indicaremos que sí deseamos establecer la referencia y una vez los cilindros estén totalmente retraídos pulsamos el interruptor situado junto a los conectores de alimentación y presión, en caso contrario no es necesario ya que se mantiene la referencia que anteriormente habíamos introducido.

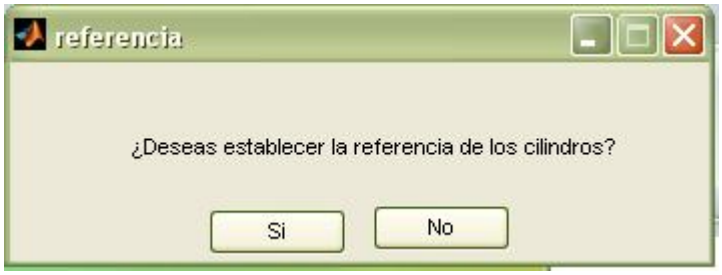


Ilustración 5. 5. Pantalla de referenciado de los cilindros.

Una vez la función referencia ha finalizado pasamos a la pantalla *menú* (Ilustración 5.6.), este es el programa que controla los movimientos del robot.

En los ejemplos de los apartados anteriores el movimiento del robot estaba predeterminado, mediante líneas de código introducidas previamente en la función *callback_PID*, y que sólo podían ser modificadas "a priori" sin que pudiésemos mover los cilindros a nuestro antojo. En cambio, ahora cada cilindro se podrá mover indicando la posición deseada mediante el uso de barras deslizantes.

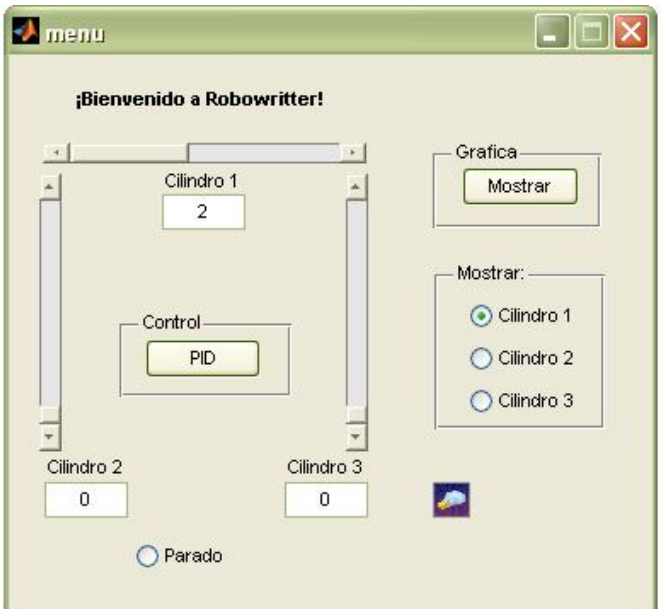


Ilustración 5. 6. Pantalla menú donde controlar los movimientos del robot.

Debajo de las barras deslizantes aparece el valor numérico que adopta dicha barra en cada momento junto al nombre del cilindro que controla.

Se denomina cilindro 1 al cilindro *DNC* y el cilindro 2 es el situado fijo formando 120º respecto al elemento de unión mientras el restante es el

denominado cilindro 3.

En el panel de control disponemos de un botón con dos posiciones posibles, por defecto aparece control PID, así el control que se realiza es de tipo PID, en caso de activarlo cambiará a control PD.

Con el botón que está situado centrado en la parte baja, hacemos que empiece el programa o, si ya se encuentra funcionando, lo paramos. Por defecto aparece con el mensaje "Parado"; si lo activamos aparecerá el mensaje "En marcha" y se encenderá la bombilla.

En el panel Mostrar seleccionamos qué cilindro queremos que aparezca representado al pulsar sobre el botón Mostrar, sólo puede haber uno activo al mismo tiempo.

Una vez queramos finalizar el uso del robot pulsamos sobre el botón que pone "En marcha" para finalizar y tras esto cerramos el programa con la X en la esquina superior derecha.

5.4.2. Generación del código de las funciones que forman la interfaz.

A continuación se muestra el código de las funciones que forman la interfaz del programa no así las funciones *PID, callback_PID, finaliza y graficas 1,2,3* ya que son muy similares a las que aparecen en el apartado *5.3.4.* salvo por pequeñas modificaciones introducidas para realizar el paso de datos entre la interfaz y estas funciones.

El código de la pantalla de presentación es el siguiente:

```
function Robowritter
clear,clc,cla,close all

%creamos figura
figdiag=figure('Units','Normalized','Number','off','Name','UPCT','Menu
bar','none','color',[0 0 0]);

%Ubicamos ejes en figura
axes('Units','Normalized','Position',[0 0 1 1]);

%incluir imagen
[x,map]=imread('Dibujo.bmp','bmp');
image(x),colormap(map),axis off, hold on

%Titulos sobre la imagen
```


```
text(50,50,'Robowritter','Fontname','Arial','Fontsize',25,'Fontangle',
'Italic','Fontweight','Bold','color',[1 1 0]);

text(50,130,'Por: Alejandro Rosillo Meseguer','Fontname','Times New
Roman','Fontsize',14,'Fontangle','Italic','Fontweight','Bold','color',
[1 1 1]);

%boton para continuar
botok=uicontrol('Style','pushbutton','Units','normalized','Position',[
.84 .03 .12 .05],'String','CONTINUAR','Callback','clear all;close
all;clc;referencia;');
```

Ahora pasamos a la pantalla encargada de situar la referencia de los cilindros:

```
function varargout = referencia(varargin)
% REFERENCIA M-file for referencia.fig
%      REFERENCIA, by itself, creates a new REFERENCIA or raises the
existing
%      singleton*.
%
%      H = REFERENCIA returns the handle to a new REFERENCIA or the
handle to
%      the existing singleton*.
%
%      REFERENCIA('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in REFERENCIA.M with the given input
arguments.
%
%      REFERENCIA('Property','Value',...) creates a new REFERENCIA or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before referencia_OpeningFunction gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to referencia_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help referencia

% Last Modified by GUIDE v2.5 11-Apr-2007 14:14:10


% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
```

```
                   'gui_OpeningFcn', @referencia_OpeningFcn, ...
                   'gui_OutputFcn',  @referencia_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before referencia is made visible.
function referencia_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to referencia (see VARARGIN)

% Choose default command line output for referencia
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes referencia wait for user response (see UIRESUME)
%uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = referencia_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

delete(handles.figure1);
disp('espere');

ao=analogoutput('nidaq',1);
set(ao,'samplerate',30);
addchannel(ao,0:1);
```


```
set(ao,'triggertype','immediate');
set(ao,'repeatoutput',inf);

dio=digitalio('nidaq',1);
addline(dio, 0:2, 0,'out');

x=4;
y=4;
putdata(ao,[x' y';x' y']);

start(ao);
pause(0.6);
stop(ao);
delete(ao);

%comprobar que el cilindro DNC este en la posicion de inicio y no la
de
%final y activacion de cable gris (si se dispone de circuitos
adaptadores)

putvalue(dio,[1 0 1]);
pause(0.2);
putvalue(dio,[1 0 0]);
stop(dio);
delete(dio);



disp('ha concluido');
menu;

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%borra la pantalla y pasa al menu
delete(handles.figure1);
menu;
```

Finalmente abrimos el *m-file* del menú y encontramos lo siguiente:

```
function varargout = menu(varargin)
% MENU M-file for menu.fig
%      MENU, by itself, creates a new MENU or raises the existing
%      singleton*.
%
%      H = MENU returns the handle to a new MENU or the handle to
%      the existing singleton*.
%
%      MENU('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in MENU.M with the given input
arguments.
%
```

```
%      MENU('Property','Value',...) creates a new MENU or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before menu_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to menu_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help menu

% Last Modified by GUIDE v2.5 05-Jul-2007 16:15:21

%declaracion de variables
global a;
global PID;


% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @menu_OpeningFcn, ...
                   'gui_OutputFcn',  @menu_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before menu is made visible.
function menu_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to menu (see VARARGIN)

%Aqui colocamos los valores que aparecen por defecto al iniciar y que
luego
%podran ser cambiados
PID=0;
handles.PID=PID;
```


```
handles.mostrargrafica=1;

    [x,map]=imread('ReadyOff.bmp');
    image(x),colormap(map),axis off

% Choose default command line output for menu
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes menu wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = menu_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure

varargout{1} = handles.output;
%varargout{2} = handles.b;

guidata(hObject, handles);


% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
%handles=get(hObject,'Value');

%Escribe el valor de Slider en dinamictext
set(handles.edit1,'String',get(hObject,'Value'));

guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
```

```
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function slider2_Callback(hObject, eventdata, handles)
% hObject    handle to slider2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider


%Escribe el valor de Slider en dinamictext
set(handles.edit2,'String',get(hObject,'Value'));

guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function slider3_Callback(hObject, eventdata, handles)
% hObject    handle to slider3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider


%Escribe el valor de Slider en dinamictext
set(handles.edit3,'String',get(hObject,'Value'));
%handles.a=handles.slider3;
guidata(hObject, handles);

%handles.output = hObject;
```


```
% --- Executes during object creation, after setting all properties.
function slider3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1
as a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2
as a double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3
as a double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton3

%situa los valores de las barras deslizantes en el vector a que  luego
se
%pasa a la función PID y callback_PID
handles.a(1)=handles.slider3;
handles.a(2)=handles.slider1;
handles.a(3)=handles.slider2;

handles.radio=get(hObject,'Value');

%Ocurre si se activa el boton
if handles.radio==1
    [x,map]=imread('ReadyOn.bmp');
    image(x),colormap(map),axis off
```


```
    set(handles.radiobutton3,'String','En marcha');

%ejecuta la funcion PID dandole los valores de posicion y el valor de
la
%variable PID que indica que valor de PID se utiliza (este vaslor esta
%dentro de la funcion callback_PID
[ai, ao, dio] = PID(handles.a,handles.PID);
ai.SamplesAcquiredFcn = {@callback_PID, ao,
dio,handles.a,handles.PID};

tic

handles.ai=ai;
handles.ao=ao;
handles.dio=dio;

%Ocurre si se desactiva el boton
elseif handles.radio==0
    [x,map]=imread('ReadyOff.bmp');
    image(x),colormap(map),axis off
    set(handles.radiobutton3,'String','Parado');

    %Llama a la función finaliza que finaliza los objetos ai,ao,dio
    finaliza(handles.ai, handles.ao, handles.dio);
    toc

end
guidata(hObject,handles);

% --- Executes on button press in togglebutton3.
function togglebutton3_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton3
handles.toggle=get(hObject,'Value');

%da el valor 0 o 1 a la variable PID
if handles.toggle==1
    set(handles.togglebutton3,'String','PD');

PID=1;

elseif handles.toggle==0
     set(handles.togglebutton3,'String','PID');

PID=0;
end

handles.PID=PID;
guidata(hObject,handles);
```

```
% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Llama a la función grafica selecionada por uipanel4
mostrargrafica=handles.mostrargrafica;
if mostrargrafica==1
    graficas1(handles.ai,handles.ao);
elseif mostrargrafica==2
    graficas2(handles.ai,handles.ao);
elseif mostrargrafica==3
    graficas3(handles.ai,handles.ao);
end

% --------------------------------------------------------------------
function uipanel4_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to uipanel4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%seleciona que grafica se mostrara al pulsar el boton de Mostrar
if (hObject==handles.uno)
    mostrargrafica=1;
elseif (hObject==handles.dos)
    mostrargrafica=2;
elseif (hObject==handles.tres)
    mostrargrafica=3;
else
    mostrargrafica=0;
end

handles.mostrargrafica=mostrargrafica;
guidata(hObject,handles);
```

Como se puede ver dentro de las funciones aparecen ciertos comentarios en inglés, estos son generados junto con las funciones por el editor *GUIDE* y en algunos casos ayudan a comprender el funcionamiento de cada función, o como hacer para utilizar datos de esa función o pasar datos entre funciones.

Dentro de las funciones que implementan el funcionamiento de los distintos elementos del menú existe líneas de código o llamadas a otras funciones con vistas al correcto funcionamiento del programa.

Todas las funciones que forman el programa, tanto las aquí mostradas como las que no lo están, se encuentran en la carpeta “Robowritter”.

La información necesaria acerca de la creación de interfaces gráfica esta disponible en la ayuda de Matlab [15].

5.5. Compilación del programa.

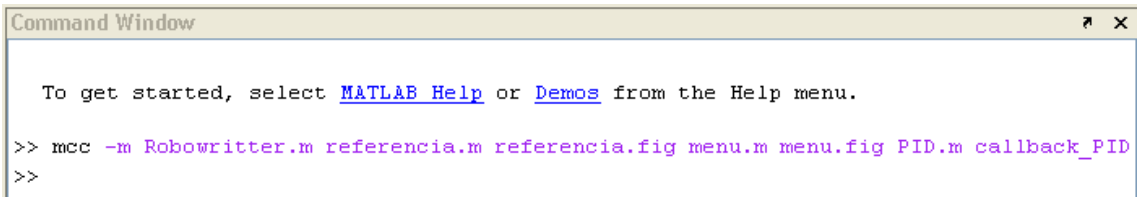
Para obtener el programa compilado de forma que no sea necesario disponer del programa *Matlab* para su ejecución escribimos lo siguiente en el *command window* de *Matlab*:

>> mcc -m Robowritter.m referencia.m referencia.fig menu.m menu.fig PID.m callback_PID.m finaliza.m graficas1.m graficas2.m graficas3.m

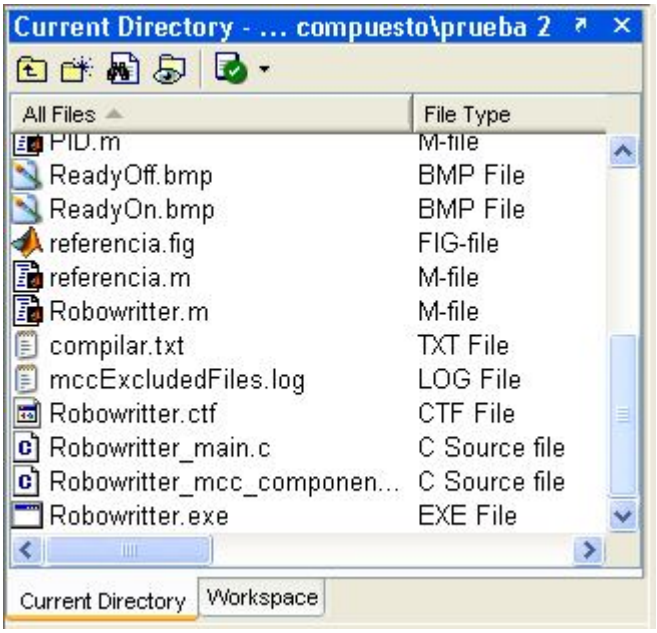
mcc invoca al compilador de Matlab

con –m generamos una aplicación en C

Después de invocar al compilador e indicarle que queremos que el programa sea generado en lenguaje C escribiremos el nombre de todas las funciones y figuras que forman parte del programa. Nótese que no he hecho uso de ningún Script, esto se debe a que el compilador solo convierte las funciones y las figuras y de existir algún Script en nuestro programa se detendría la compilación.



Tras ejecutar estas líneas empieza la compilación. Seguimos las instrucciones, indicamos el compilador deseado de la lista que nos aparece y surgirán nuevos archivos en la carpeta donde teníamos el programa. Uno de estos nuevos ficheros es un ejecutable, lo abrimos y comienza a extraer archivos; esto puede resultar un poco lento en algunos ordenadores e incluso puede parecer que el ordenador se haya colgado, un poco de paciencia.



Después de abrir el fichero ejecutable y terminar la extracción de archivos nos aparece una nueva carpeta denominada *Robowritter_mcr* mientras se ejecuta el programa. Las próximas veces cuando abramos el ejecutable el ordenador no tendrá que extraer los ficheros contenidos en esa carpeta por lo que todo irá mas rápido; ahora tenemos el programa listo para ser usado en cualquier ordenador, incluso aunque no disponga del programa *Matlab.* El programa compilado se encuentra contenido en la carpeta *"Robowritter compilado".*

Cualquier modificación realizada a las funciones aquí presentes, producirá cambios en el programa sin necesidad de compilarlo de nuevo.