REPÚBLICA DEL ECUADOR

UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

ESCUELA DE ELECTRÓNICA Y TELECOMUNICACIONES

MANUAL DE INTERFAZ GRÁFICA DE USUARIO EN MATLAB

AUTOR: Diego Orlando Barragán Guerrero

diegokillemall@yahoo.com

dobarragan@utpl.edu.ec

ÍNDICE

"…lo que uno es para sí lo que le acompaña en la soledad y que nadie puede darle o quitarle, es más esencial que todo lo que posee o lo que pueda ser a los ojos de los demás"

Arthur Schopenhauer

GUIDE

INTERFAZ GRÁFICA DE USUARIO EN MATLAB

INTRODUCCIÓN

GUIDE es un entorno de programación visual disponible en MATLAB para realizar y ejecutar programas que necesiten ingreso continuo de datos. Tiene las características básicas de todos los programas visuales como Visual Basic o Visual C++.

INICIO

Para iniciar nuestro proyecto, lo podemos hacer de dos maneras:

- Ejecutando la siguiente instrucción en la ventana de comandos:

```
>> guide
```

- Haciendo un click en el ícono que muestra la figura:

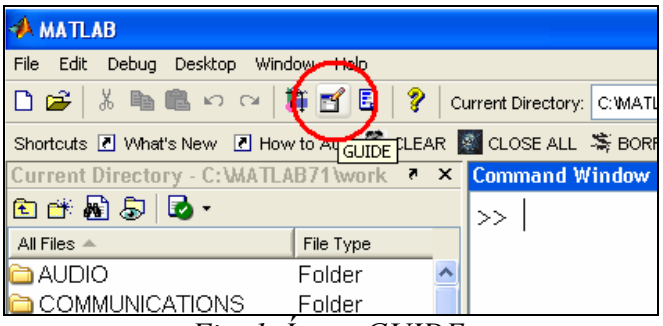


Fig. 1. Ícono GUIDE.

Se presenta el siguiente cuadro de diálogo:

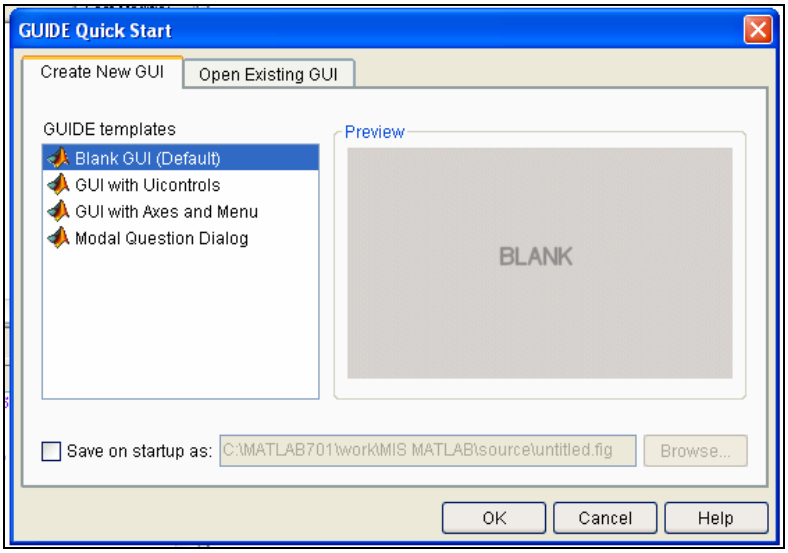


Fig. 2. Ventana de inicio de GUI.

Se presentan las siguientes opciones:

a) **Blank GUI (Default)**
La opción de interfaz gráfica de usuario en blanco (viene predeterminada), nos presenta un formulario nuevo, en el cual podemos diseñar nuestro programa.

b) **GUI with Uicontrols**
Esta opción presenta un ejemplo en el cual se calcula la masa, dada la densidad y el volumen, en alguno de los dos sistemas de unidades. Podemos ejecutar este ejemplo y obtener resultados.

c) **GUI with Axes and Menu**
Esta opción es otro ejemplo el cual contiene el menú File con las opciones Open, Print y Close. En el formulario tiene un *Popup menu*, un *push button* y un objeto *Axes*, podemos ejecutar el programa eligiendo alguna de las seis opciones que se encuentran en el menú despegable y haciendo click en el botón de comando.

d) **Modal Question Dialog**
Con esta opción se muestra en la pantalla un cuadro de diálogo común, el cual consta de una pequeña imagen, una etiqueta y dos botones *Yes* y *No*, dependiendo del botón que se presione, el GUI retorna el texto seleccionado (la cadena de caracteres 'Yes' o 'No').

Elegimos la primera opción, *Blank GUI,* y tenemos:

Fig. 3. Entorno de diseño de GUI

La interfaz gráfica cuenta con las siguientes herramientas:

	Alinear objetos.
	Editor de menú.
	Editor de orden de etiqueta.
	Editor del M-file.
	Propiedades de objetos.
	Navegador de objetos.
	Grabar y ejecutar (ctrl. + T).

Para obtener la etiqueta de cada elemento de la paleta de componentes ejecutamos: *File>>Preferentes* y seleccionamos *Show names in component palette.* Tenemos la siguiente presentación:

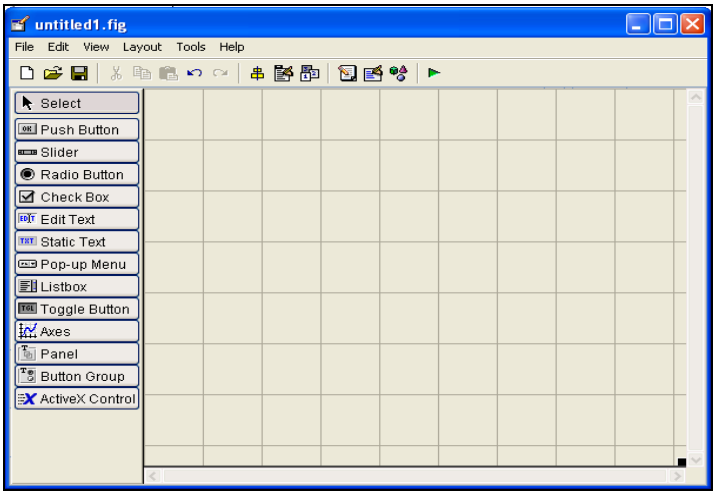


Fig. 4. Entorno de diseño: componentes etiquetados.

La siguiente tabla muestra una descripción de los componentes:

Control	***Valor de estilo***	***Descripción***
Check box	'checkbox'	Indica el estado de una opción o atributo
Editable Text	'edit'	Caja para editar texto
Pop-up menu	'popupmenu'	Provee una lista de opciones
List Box	'listbox'	Muestra una lista deslizable
Push Button	'pushbutton'	Invoca un evento inmediatamente
Radio Button	'radio'	Indica una opción que puede ser seleccionada
Toggle Button	'togglebutton'	Solo dos estados, "on" o "off"
Slider	'slider'	Usado para representar un rango de valores
Static Text	'text'	Muestra un string de texto en una caja
Panel button		Agrupa botones como un grupo
Button Group		Permite exclusividad de selección con los radio button

PROPIEDADES DE LOS COMPONENTES

Cada uno de los elementos de GUI, tiene un conjunto de opciones que podemos acceder con click derecho.

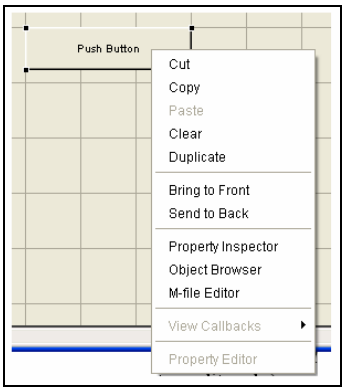


Fig. 5. Opciones del componente.

La opción *Property Inspector* nos permite personalizar cada elemento.

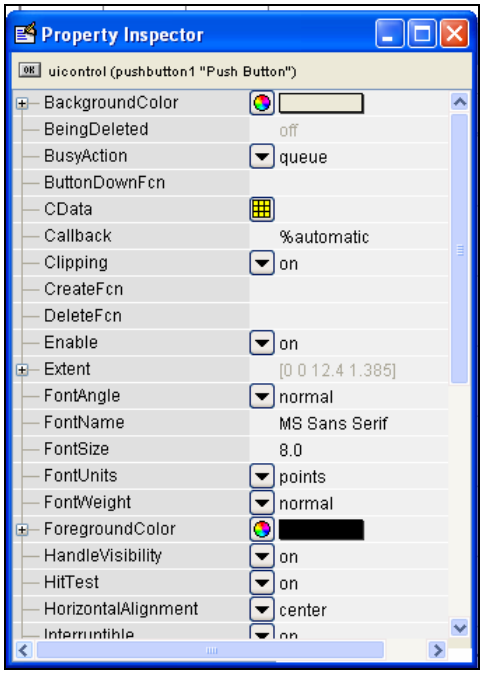


Fig. 6. Entorno Property Inspector.
Permite ver y editar las propiedades de un objeto.

Al hacer click derecho en el elemento ubicado en el área de diseño, una de las opciones más importantes es *View Callbacks*, la cual, al ejecutarla, abre el archivo *.m* asociado a nuestro diseño y nos posiciona en la parte del programa que corresponde a la subrutina que se ejecutará cuando se realice una determinada acción sobre el elemento que estamos editando.

Por ejemplo, al ejecutar *View Callbacks>>Callbacks* en el *Push Button*, nos ubicaremos en la parte del programa:

```
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)
```

FUNCIONAMIENTO DE UNA APLICACIÓN GUI

Una aplicación GUIDE consta de dos archivos: *.m* y *.fig*. El archivo *.m* es el que contiene el código con las correspondencias de los botones de control de la interfaz y el archivo *.fig* contiene los elementos gráficos.

Cada vez que se adicione un nuevo elemento en la interfaz gráfica, se genera automáticamente código en el archivo *.m*.

Para ejecutar una Interfaz Gráfica, si la hemos etiquetado con el nombre *curso.fig*, simplemente ejecutamos en la ventana de comandos >> *curso*. O haciendo click derecho en el m-file y seleccionando la opción *RUN*.

MANEJO DE DATOS ENTRE LOS ELEMENTOS DE LA APLICACIÓN Y EL ARCHIVO .M

Todos los valores de las propiedades de los elementos (color, valor, posición, string…) y los valores de las variables transitorias del programa se almacenan en una estructura, los cuales son accedidos mediante un único y mismo *identificador* para todos éstos. Tomando el programa listado anteriormente, el identificador se asigna en:

```
handles.output = hObject;
```

handles, es nuestro identificador a los datos de la aplicación. Esta definición de identificador es salvada con la siguiente instrucción:

```
guidata(hObject, handles);
```

guidata, es la sentencia para salvar los datos de la aplicación.

Aviso: *guidata* es la función que guarda las variables y propiedades de los elementos en la estructura de datos de la aplicación, por lo tanto, como regla general, en cada subrutina se debe escribir en la última línea lo siguiente:

```
guidata(hObject,handles);
```

Esta sentencia nos garantiza que cualquier cambio o asignación de propiedades o variables quede almacenado.

Por ejemplo, si dentro de una subrutina una operación dio como resultado una variable *diego* para poder utilizarla desde el programa u otra subrutina debemos salvarla de la siguiente manera:

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

La primera línea crea la variable *diego* a la estructura de datos de la aplicación apuntada por *handles* y la segunda graba el valor.

SENTENCIAS GET Y SET

La asignación u obtención de valores de los componentes se realiza mediante las sentencias *get* y *set*. Por ejemplo si queremos que la variable *utpl* tenga el valor del *Slider* escribimos:

```
utpl= get(handles.slider1,'Value');
```

Notar que siempre se obtienen los datos a través de los identificadores *handles*.

Para asignar el valor a la variable *utpl* al *statictext* etiquetada como *text1* escribimos:

```
set(handles.text1,'String',utpl);%Escribe el valor del Slider...
                                 %en static-text
```

EJEMPLOS[1]

PRESENTACIÓN DE NUESTRO PROGRAMA

Fig. 7. Presentación del programa.

Para la presentación de nuestro programa, vamos a programar una pantalla donde podemos colocar el tema de nuestro diseño, nombre del programador, imagen de fondo…

Para esto, tenemos el siguiente código (copiar y pegar en un m-file):

```
function presentacion
%Autor: Diego Barragán Guerrero
%Estudiante de Electrónica y Telecomunicaciones
%*************************************************************
% presentación: función que presenta la pantalla de presentación
%*************************************************************
clear,clc,cla,close all

%Creamos figura
figdiag=figure('Units','Normalized',...
```

[1] Para nuestro curso, creamos en `C:\MATLAB71\work` la carpeta MIS_MATLAB, donde almacenaremos todos nuestros programas.

```
                'Position',[0.0725 0.0725 0.57 0.57],... %Tamaño de la
presentación
                'Number','off',...
                'Name','Electrónica y Telecomunicaciones', ...
                'Menubar','none', ...
                'color',[0 0 0]);

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

%Incluir imagen
%Importamos imagen *.jpg,junto con su mapa de colores
[x,map]=imread('circuit.jpg','jpg');
%Representamos imagen en figura, con su mapa de colores
image(x),colormap(map),axis off,hold on

%Títulos sobre imagen
%Título
text(50,50,'Presentación del
Programa','Fontname','Arial','Fontsize',25,'Fontangle','Italic', ...
'Fontweight','Bold','color',[1 1 0]);

%Nombre del programador
text(50,130,'Por: Diego Barragán Guerrero','Fontname', ...
'Comic Sans MS','Fontangle','Italic','Fontweight','Bold', ...
'Fontsize',14,'color',[1 1 1]);

%Botón Continuar
botok=uicontrol('Style','pushbutton', ...
    'Units','normalized', ...
   'Position',[.84 .03 .12 .05], ...
    'String','CONTINUAR',...
    'Callback','clear all; close all;clc; GUI;'); %GUI es el nombre
del siguiente programa.
```

Las sentencias luego del comentario `%Botón Continuar`, en la línea `'Callback','clear all; close all;clc; GUI;');` se reemplaza GUI por el nombre de nuestro programa diseñado con interfaz de usuario.

Para ejecutarlo, presionamos la tecla F5.

Otra manera de presentar nuestro programa es con la siguiente función:

```
function presen(filename,varargin)
%presen(filename,varargin)
%filename es el nombre de una imagen y su extensión.
%varargin es el tiempo en milisegundos.
%Ejemplo:
% presen('portada.jpg',2000)
if nargin ==1
    I = imread(filename);
    time = 4000;
elseif (nargin == 2)&(ischar(varargin{1}))
    fmt = varargin{1};
    I = imread(filename,fmt);
    time = 4000;
elseif (nargin == 2)&(isnumeric(varargin{1}))
```

```
    I = imread(filename);
    time = varargin{1};
elseif nargin == 3    fmt = varargin{1};
    I = imread(filename,fmt);
    time = varargin{2};
    if (~isnumeric(time))| (length(time)~=1)
        error('ERROR: TIME debe ser un valor numérico en seg.');
    end
else
    error('ERROR: demasiados datos entrada!');
end
judasImage = im2java(I);
win = javax.swing.JWindow;
icon = javax.swing.ImageIcon(judasImage);
label = javax.swing.JLabel(icon);
win.getContentPane.add(label);
screenSize = win.getToolkit.getScreenSize;
screenHeight = screenSize.height;
screenWidth = screenSize.width;
imgHeight = icon.getIconHeight;
imgWidth = icon.getIconWidth;
win.setLocation((screenWidth-imgWidth)/2,(screenHeight-imgHeight)/2);
win.pack
win.show
tic;
while toc < time/1000
end
win.dispose()
```

No olvidar que las funciones se guardan con el mismo nombre.

PROGRAMA *SUMADORA*

Con este ejemplo, se pretende mostrar el uso de *pushbutton*, *static text* y *Edit text*, así como insertar una imagen de fondo a nuestro diseño.

Fig. 8. Entorno del programa sumadora.

Primero, ejecutamos `>> guide` en la ventana de comandos, y presionamos *Enter*.

Seleccionamos **Blank GUI (default)** y presionamos OK.

Insertamos los componentes que muestra la siguiente figura:

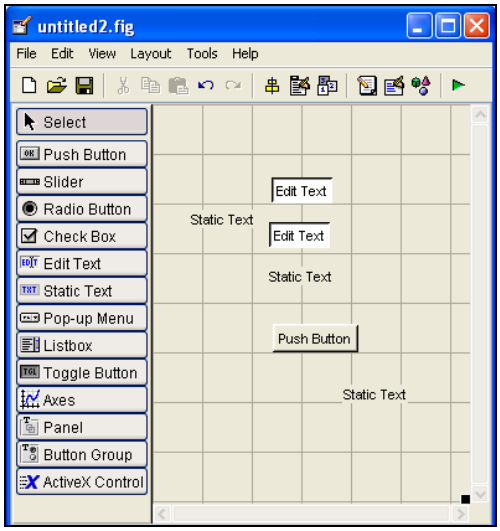


Fig. 9. Componentes del programa sumadora.

Haciendo doble-click en cada componente, accedemos a configurar las propiedades de cada elemento. Iniciando en *pushbutton*, configuramos:

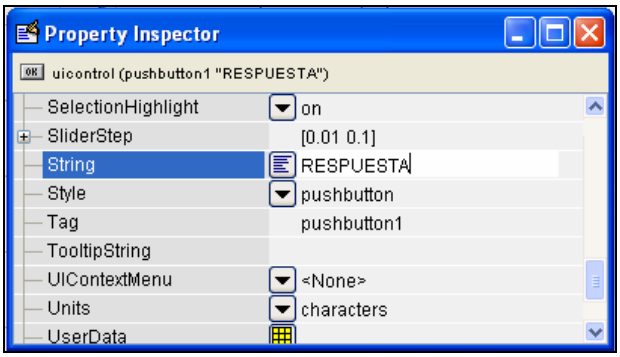


Fig. 10. Editar componente.

Podemos cambiar el nombre con el que aparecerá la función del *pushbutton* en el m-file, simplemente editando el campo *Tag*.

Continuamos editando los demás componentes, hasta llegar a tener una presentación semejante a la figura del inicio de este ejemplo (de la imagen de fondo nos ocuparemos al final).

Ejecutando Ctrl+T o presionando  , guardamos[2] nuestro programa con el nombre *Sumadora* en la carpeta MIS_MATLAB. A continuación en la ventana *Current Directory* aparecen el archivo *Sumadora.fig* y *Sumadora.m*.

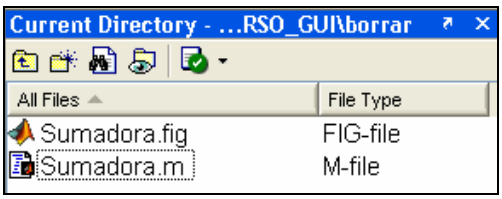


Fig. 11. Archivos en Current Directory.

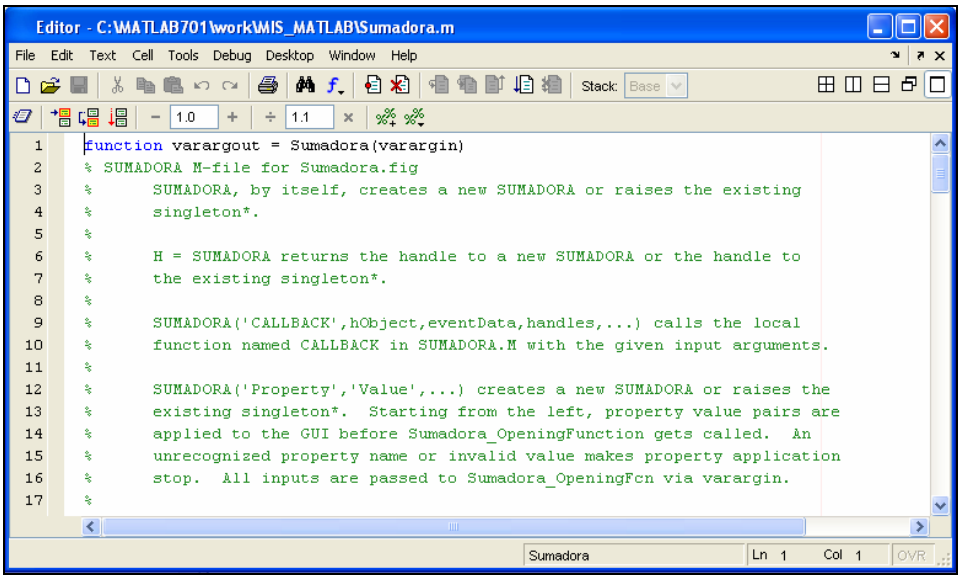


Fig. 12. M-file asociado.

[2] Recuerde que Matlab hace diferencia entre mayúsculas y minúsculas. Asimismo, nunca empiece el nombre de un archivo con números ni guiones bajos, no use tildes ni la letra ñ. Para separar dos palabras use guión bajo (ejp: mi_programa).

Para iniciar a editar nuestro m-file, llegamos a cada función con el ícono *Show functions*  , como muestra la siguiente figura:

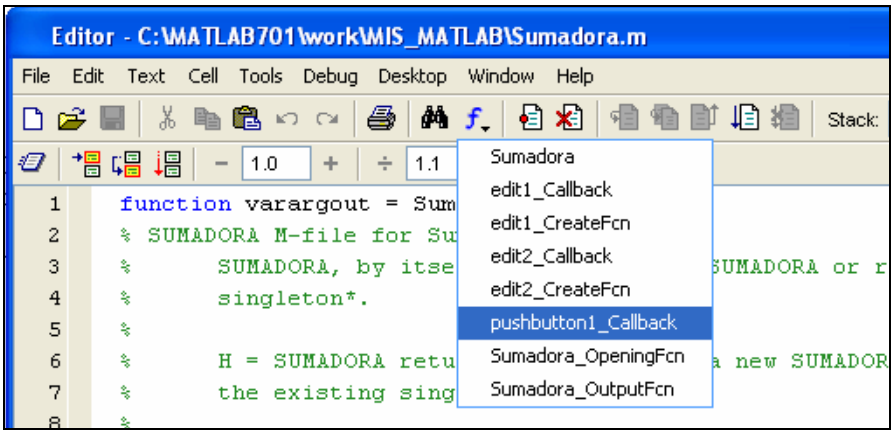


Fig. 13. Botón Show Functions.

Cada uno de los elementos añadidos en nuestro diseño como *pushbutton, edit text, static text* tienen una función asociada en nuestro m-file. Así, al añadir *pushbutton,* tenemos el siguiente código:

```
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)
```

Cada *edit text* tendrá el siguiente código:

```
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
```

Static text, no posee función asociada, pero sí una dirección asociada, que la podemos utilizar para escribir los resultados. Para saber cuál es esta dirección, haciendo doble-click en este componente, la ubicamos en la etiqueta *Tag*, como lo muestra la siguiente figura:

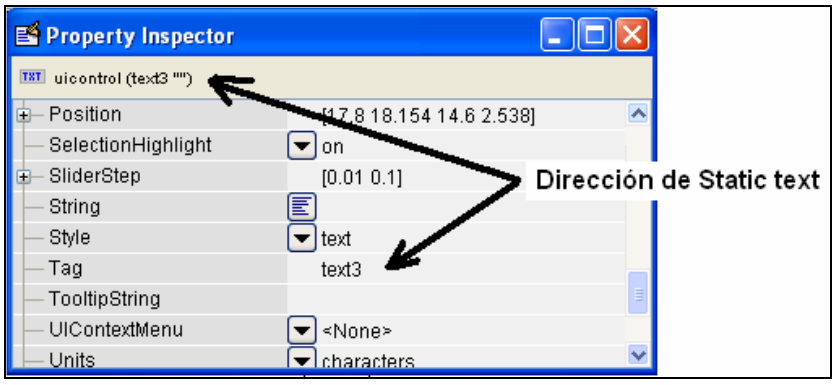


Fig. 14. Dirección de Static text.

Aquí empieza lo bueno. Justo debajo de `function edit1_Callback(hObject, eventdata, handles)`, y de los comentarios correspondientes, escribimos el siguiente código:

```
Val=get(hObject,'String'); %Almacenar valor ingresado
NewVal = str2double(Val);  %Transformar a formato double
handles.edit1=NewVal;            %Almacenar en identificador
guidata(hObject,handles);        %Salvar datos de la aplicación
```

Recuérdese que la instrucción *get* la usamos para obtener datos ingresados por el usuario. Así, la línea `Val=get(hObject,'String')` almacena en `Val` el valor ingresado en formato *String*. La sentencia `NewVal = str2double(Val)` realiza la transformación de *string* a *double*, o de palabra a número. La sentencia `handles.edit1=NewVal` almacena `NewVal` en el identificador `handles.edit1`. Por último, salvamos los datos de la aplicación con la sentencia `guidata(hObject,handles)`.

Repetimos las mismas sentencias justo debajo de `function edit2_Callback(hObject, eventdata, handles)`, pero esta vez usando el identificador `handles.edit2=NewVal`. Tendremos las siguientes sentencias.

```
Val=get(hObject,'String'); %Almacenar valor ingresado
NewVal = str2double(Val);  %Transformar a formato double
handles.edit2=NewVal;            %Almacenar en identificador
guidata(hObject,handles);        %Salvar datos de la aplicación
```

Hasta el momento tenemos los dos sumandos almacenados en los identificadores *handles.edit1* y *handles.edit2*. Como nuestro resultado se muestra al presionar el botón RESPUESTA, es momento de editar la función correspondiente a *pushbutton*.

Debajo de `function pushbutton1_Callback(hObject, eventdata, handles)`, y de los comentarios correspondientes, editamos el siguiente código:

```
A=handles.edit1;
B=handles.edit2;
suma=A+B;
set(handles.text3,'String',suma);
```

Las tres primeras sentencias son por demás obvias. Sin embargo, la cuarta línea contiene la instrucción *set*, con la cual establecemos un valor (*string*) al componente *Static text3*, con el identificador *handles.text3*.

Bien, hasta aquí ya tenemos nuestra sumadora. Ejecutamos el programa con F5.

La imagen de fondo puede ser cualquier imagen jpg. Añadimos a nuestro diseño el componente *axes* y en el campo *Tag* de *Property Inspector* lo editamos con el nombre *background*. El siguiente código, colocado en la función de apertura (`function Sumadora_OpeningFcn`), carga la imagen de fondo:

```
function Sumadora_OpeningFcn(hObject, eventdata, handles, varargin)
%Colocar Imagen de fondo
background = imread('background.jpg'); %Leer imagen
axes(handles.background);              %Carga la imagen en background
```

```
axis off;
imshow(background);                    %Presenta la imagen
% Choose default command line output for Sumadora
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
```

Nótese que usamos el comando *imread* para cargar la imagen e *imshow* para colocarla en *handles.background*.

Si tuviste algún problema al ejecutar el programa, aquí está el código final (`%Sin comentarios`):

```
function varargout = Sumadora(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Sumadora_OpeningFcn, ...
                   'gui_OutputFcn',  @Sumadora_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
function Sumadora_OpeningFcn(hObject, eventdata, handles, varargin)
background = imread('background.jpg'); %Leer imagen
axes(handles.background);              %Carga la imagen en background
axis off;
imshow(background);                    %Presenta la imagen
handles.output = hObject;
guidata(hObject, handles);
function varargout = Sumadora_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function pushbutton1_Callback(hObject, eventdata, handles)
A=handles.edit1;
B=handles.edit2;
ANSWER=A+B;
set(handles.text3,'String',ANSWER);
function edit1_Callback(hObject, eventdata, handles)
Val=get(hObject,'String'); %Almacenar valor ingresado
NewVal = str2double(Val);  %Transformar a formato double
handles.edit1=NewVal;            %Almacenar en identificador
guidata(hObject,handles);        %Salvar datos de la aplicación
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit2_Callback(hObject, eventdata, handles)
Val=get(hObject,'String');
NewVal = str2double(Val);
handles.edit2=NewVal;
guidata(hObject,handles);
function edit2_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Otra manera de realizar el programa es colocar todo el código dentro de la función del *pushbutton*, de la siguiente manera.

```
function pushbutton1_Callback(hObject, eventdata, handles)
A= str2double(get(handles.edit1, 'String'));
B= str2double(get(handles.edit2, 'String'));
suma=A+B;
set(handles.text3,'String',suma);
```

PROGRAMA *SUMADORA_2*

Con este programa se pretende mostrar la ejecución de cálculos con el simple hecho de ingresar datos en un *edit-text*, o sea sin necesidad de un *push-button*.

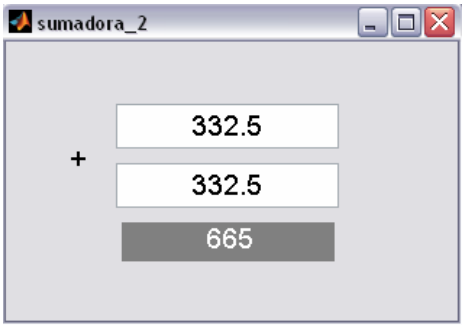


Fig. 15. Entorno sumadora_2

Lo primero que realizamos es editar en el *Property Inspector* el campo *Tag* del primer *edit-text* con *uno* y *dos* para el segundo. El campo *Tag* para mostrar el resultado se etiqueta como *resp*.

Con un click derecho en el primer *edit-text* nos ubicamos en *View Callbacks→Callback*. Esto nos ubica en la parte del M-file correspondiente para este *edit-text*. El código queda de la siguiente manera:

```
function uno_Callback(hObject, eventdata, handles)
sum1=str2double(get(hObject,'String'));
sum2=str2double(get(handles.dos,'String'));
if isnan(sum1)
    errordlg('El valor debe ser numérico','ERROR')
    set(handles.uno,'String',0);
    sum1=0;
end
if isnan(sum2)
    errordlg('El valor debe ser numérico','ERROR')
    set(handles.dos,'String',0);
    sum2=0;
end
suma=sum1+sum2;
set(handles.resp,'String',suma);
guidata(hObject, handles);
```

Como se puede ver, podemos añadir la *detección y corrección de errores* de los datos ingresado. La sentencia `isnan(sum1)` funciona como lo muestra el siguiente código:

```
>> isnan(str2double('a'))
ans =
     1
>> isnan(str2double('0'))
ans =
     0
```

De tal manera, la sentencia `if isnan(sum1)`, en caso de dar un 1, presentará un cuadro de error (estos mensajes de usuario se ven más adelante) y corrige el valor

asignando 0 al sumando donde se ha ingresado el error con la sentencia `set(handles.dos,'String',0); sum2=0.`

La programación del otro *edit-text* queda como sigue:

```
function dos_Callback(hObject, eventdata, handles)
sum2=str2double(get(hObject,'String'));
sum1=str2double(get(handles.uno,'String'));
if isnan(sum1)
    errordlg('El valor debe ser numérico','ERROR')
    set(handles.uno,'String',0);
    sum1=0;
end
if isnan(sum2)
    errordlg('El valor debe ser numérico','ERROR')
    set(handles.dos,'String',0);
    sum2=0;
end
suma=sum1+sum2;
set(handles.resp,'String',suma);
guidata(hObject, handles);
```

PROGRAMA *CALCULADORA*

Con este ejemplo exploraremos los comandos `strcat` (*string concatenation*) y `eval`. Asimismo, usaremos un cuadro mensaje de diálogo y la herramienta *Align Objects*.

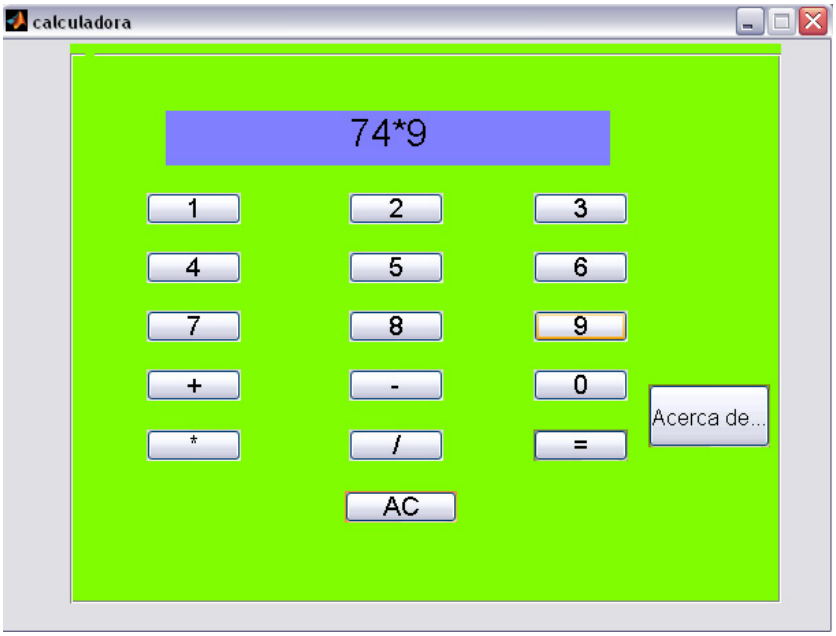


Fig. 16. Calculadora sencilla.

Abrimos un nuevo GUI y colocamos cada uno de los componentes de la figura 16. En primer lugar añadimos un *panel* y luego arrastramos los demás elementos dentro de este componente: 17 *pushbutton* y un *static text*.

Asimismo, es necesario para este ejemplo usar la herramienta de alineación de objetos. La figura 17 muestra el entrono de *Align Objects*.

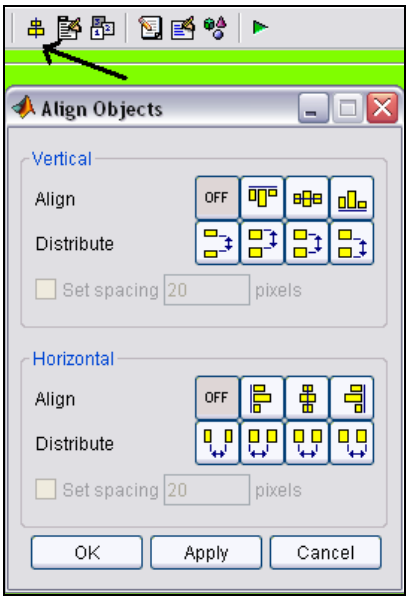


Fig. 17. Herramienta para ordenar objetos.

El comando `strcat` (*string concatenation*) une o concatena varios strings. Por ejemplo:

```
>>d='Diego';
e='_';
b='Orlando';
strcat(d,e,b)
ans =
Diego_Orlando
```

El comando `eval` evalúa y ejecuta la operación dada en formato *string*. Por ejemplo:

```
>>eval('2+3+4')
ans =
     9

>>eval('sqrt(144)')
ans =
    12
```

Hagamos un breve repaso de *pushbutton*. En un GUI en blanco, colocamos un par de *pushbutton*.

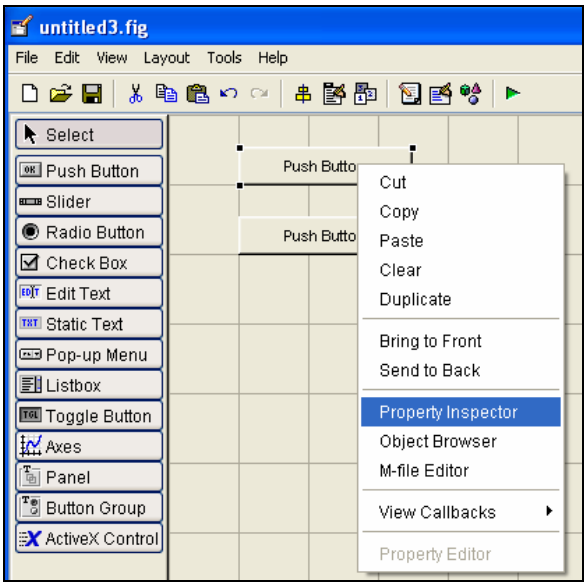


Fig. 18. Push Buttons.

Grabamos nuestro GUI de ejemplo con el nombre *eliminar*. Con click derecho, nos ubicamos en la opción que muestra la siguiente figura:

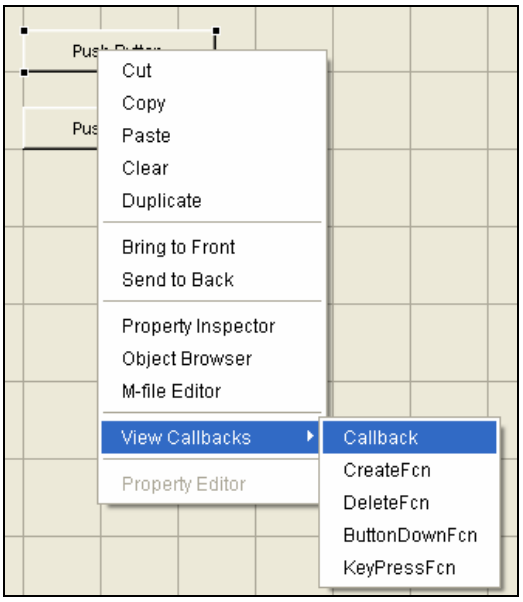


Fig. 19. Ruta View Callbacks>>Callback

Inmediatamente, esto nos ubica en la parte del m-file correspondiente a la subrutina resaltada del *pushbutton*.

```
% --- 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)
```

En *Property Editor* podemos etiquetar el nombre con el que aparecerá nuestro *pushbutton* en le m-file. En el campo *Tag*, editamos el nombre *uno*, para el primer *pushbutton* y *dos* para el segundo *pushbutton*. Nuevamente nos ubicamos en *View Callbacks>Callbacks* y podemos notar el cambio de etiqueta en el m-file.

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

Gracias a esta nueva etiqueta, en un diseño que contenga muchos botones (como el caso de nuestra calculadora) podemos ubicarnos mejor en la programación del m-file.

Ahora es momento de etiquetar cada componente de este diseño.

Al ejecutar *Show Functions*, tendremos la siguiente presentación:

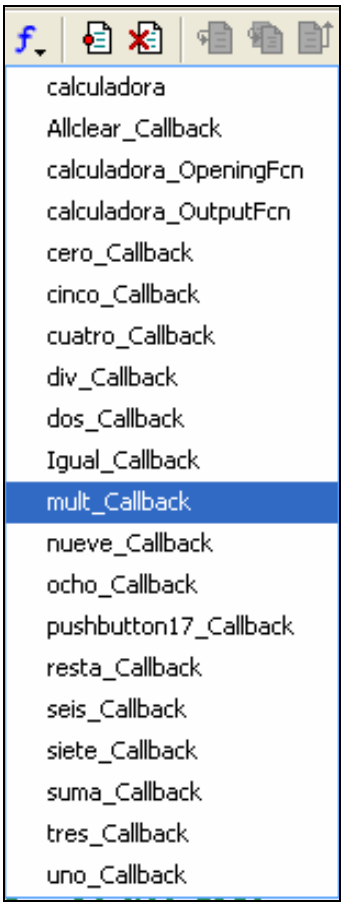


Fig. 20. Show Function.

Es momento de empezar a editar nuestro m-file. Empezando en *uno_Callback*, escribimos el siguiente código:

```
function uno_Callback(hObject, eventdata, handles)
textString=get(handles.text1,'String');
textString=strcat(textString,'1');
set(handles.text1,'String',textString)
```

Repetimos el mismo código para los demás botones, excepto para el botón IGUAL, donde debemos ejecutar el siguiente código:

```
function Igual_Callback(hObject, eventdata, handles)
%Este es el igual
textString=get(handles.text1,'String');
textString=eval(textString);
set(handles.text1,'String',textString)
```

Nótese el uso de los comandos `strcat` y `eval`.

El botón AC, etiquetado como `Allclear`, tendrá las siguientes sentencias:

```
ini=char(' ');
set(handles.text1,'String',ini);
```

Para editar el cuadro de diálogo, debajo de `function INFO_Callback(hObject, eventdata, handles)`, escribimos la siguiente sentencia:

```
msgbox('Calculadora Sencilla. Por: Diego Barragán G','Acerca de...');
```

Al ejecutar nuestro programa, tendremos el siguiente resultado:

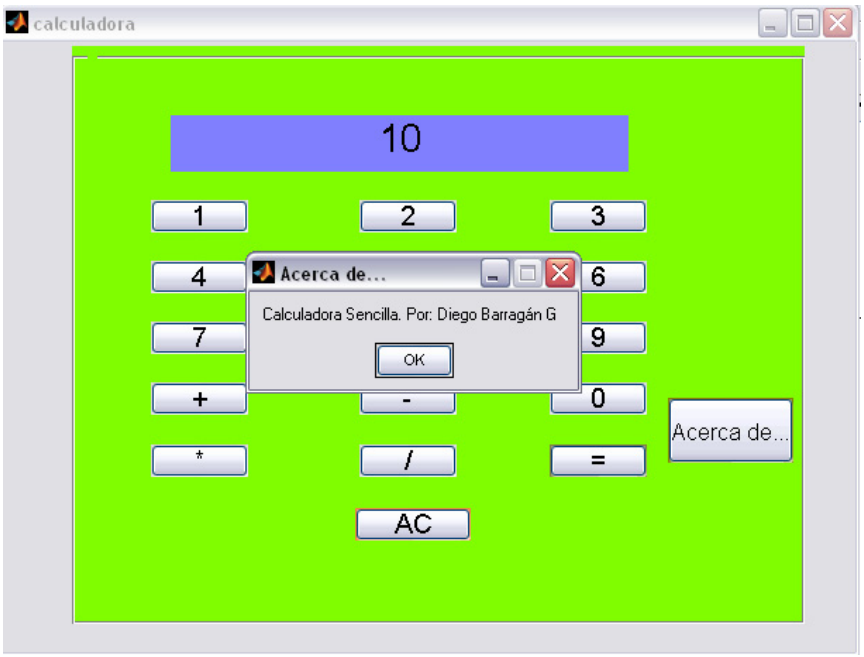


Fig. 21. Entorno del programa "calculadora".

Esto sería todo en nuestra *calculadora sencilla*. Notemos que podemos crear alguno botones más, como el punto decimal, raíz cuadrada (sqrt)…

Te copio con el código final del m-file:

```
function varargout = calculadora(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @calculadora_OpeningFcn, ...
                   'gui_OutputFcn',  @calculadora_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 calculadora is made visible.
function calculadora_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = calculadora_OutputFcn(hObject, eventdata,
handles)
varargout{1} = handles.output;

% --- Executes on button press in uno.
function uno_Callback(hObject, eventdata, handles)
textString=get(handles.text1,'String');
textString=strcat(textString,'1');
set(handles.text1,'String',textString)

% --- Executes on button press in dos.
function dos_Callback(hObject, eventdata, handles)
textString=get(handles.text1,'String');
textString=strcat(textString,'2');
set(handles.text1,'String',textString)

% --- Executes on button press in tres.
function tres_Callback(hObject, eventdata, handles)
textString=get(handles.text1,'String');
textString=strcat(textString,'3');
set(handles.text1,'String',textString)

% --- Executes on button press in cuatro.
function cuatro_Callback(hObject, eventdata, handles)
textString=get(handles.text1,'String');
textString=strcat(textString,'4');
set(handles.text1,'String',textString)

% --- Executes on button press in cinco.
function cinco_Callback(hObject, eventdata, handles)
textString=get(handles.text1,'String');
textString=strcat(textString,'5');
set(handles.text1,'String',textString)

% --- Executes on button press in seis.
function seis_Callback(hObject, eventdata, handles)
textString=get(handles.text1,'String');
textString=strcat(textString,'6');
set(handles.text1,'String',textString)

% --- Executes on button press in siete.
function siete_Callback(hObject, eventdata, handles)
textString=get(handles.text1,'String');
textString=strcat(textString,'7');
set(handles.text1,'String',textString)

% --- Executes on button press in ocho.
function ocho_Callback(hObject, eventdata, handles)
textString=get(handles.text1,'String');
textString=strcat(textString,'8');
set(handles.text1,'String',textString)

% --- Executes on button press in nueve.
function nueve_Callback(hObject, eventdata, handles)
textString=get(handles.text1,'String');
textString=strcat(textString,'9');
```

```
set(handles.text1,'String',textString)

% --- Executes on button press in suma.
function suma_Callback(hObject, eventdata, handles)
textString=get(handles.text1,'String');
textString=strcat(textString,'+');
set(handles.text1,'String',textString)

% --- Executes on button press in resta.
function resta_Callback(hObject, eventdata, handles)
textString=get(handles.text1,'String');
textString=strcat(textString,'-');
set(handles.text1,'String',textString)

% --- Executes on button press in cero.
function cero_Callback(hObject, eventdata, handles)
textString=get(handles.text1,'String');
textString=strcat(textString,'0');
set(handles.text1,'String',textString)

% --- Executes on button press in mult.
function mult_Callback(hObject, eventdata, handles)
textString=get(handles.text1,'String');
textString=strcat(textString,'*');
set(handles.text1,'String',textString)

% --- Executes on button press in div.
function div_Callback(hObject, eventdata, handles)
textString=get(handles.text1,'String');
textString=strcat(textString,'/');
set(handles.text1,'String',textString)

% --- Executes on button press in Igual.
function Igual_Callback(hObject, eventdata, handles)
%Este es el igual
textString=get(handles.text1,'String');
textString=eval(textString,'1');
set(handles.text1,'String',textString)

% --- Executes on button press in Allclear.
function Allclear_Callback(hObject, eventdata, handles)
%Limpiar pantalla
ini=char(' ');
set(handles.text1,'String',ini);
% --- Executes on button press in INFO.
function INFO_Callback(hObject, eventdata, handles)
msgbox('Calculadora Sencilla. Por: Diego Barragán G','Acerca de...');
```

MENSAJES DE USUARIO

Como vimos en el ejemplo de la sumadora_2 y la calculadora, podemos añadir un cuadro de mensaje para el usuario. Existen algunos tipos y como ejemplo vamos a crear un nueva GUI con el nombre *mensajes*. Colocamos en el mismo un panel y dentro del panel cinco *pushbutton*. En *Property Inspector* editamos los nombres como muestra la figura y editamos el campo *Tag* con los mismos nombres.

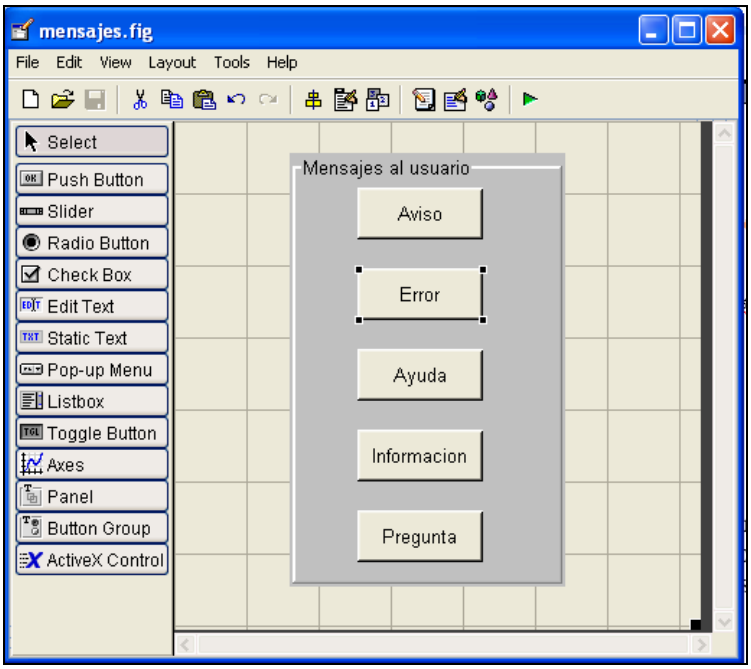


Fig. 22. Mensajes al usuario.

Las siguientes sentencias se ubican debajo de la función correspondiente:

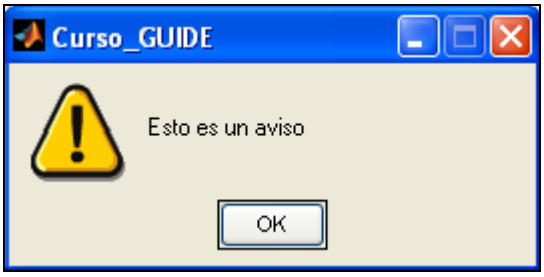
```
warndlg('Esto es un aviso','Curso_GUIDE');
errordlg('Esto es un mensaje de error',' Curso_GUIDE ');
helpdlg('Esto es una ayuda',' Curso_GUIDE ');
msgbox('Esto es un cuadro de mensaje',' Curso_GUIDE ');
questdlg('Esto es una pregunta',' Curso_GUIDE ');
```

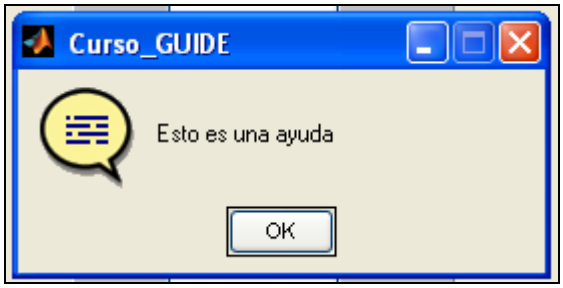
Parte de nuestro m-file queda de la siguiente forma:

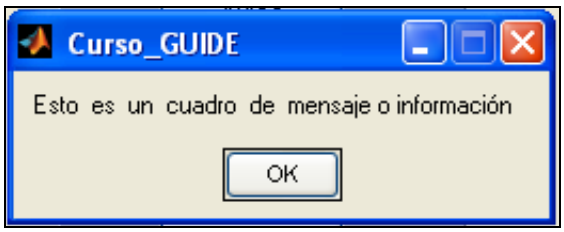
```
% --- Executes on button press in Aviso.
function Aviso_Callback(hObject, eventdata, handles)
warndlg('Esto es un aviso','Curso_GUIDE');

% --- Executes on button press in Error_c.
function Error_c_Callback(hObject, eventdata, handles)
errordlg('Esto es un mensaje de error',' Curso_GUIDE ');

% --- Executes on button press in Ayuda.
function Ayuda_Callback(hObject, eventdata, handles)
helpdlg('Esto es una ayuda',' Curso_GUIDE ');

% --- Executes on button press in informacion.
function informacion_Callback(hObject, eventdata, handles)
msgbox('Esto es un cuadro de mensaje',' Curso_GUIDE ');
```

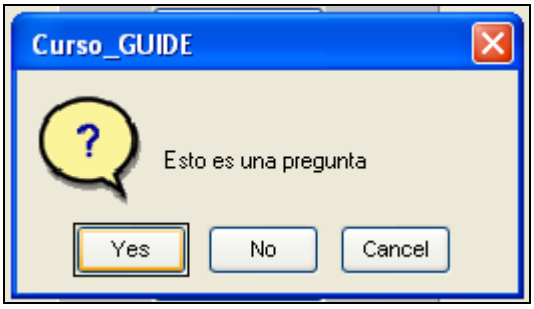
```
% --- Executes on button press in Pregunta.
function Pregunta_Callback(hObject, eventdata, handles)
questdlg('Esto es una pregunta',' Curso_GUIDE ');
```

Al presionar cada botón, tendremos los siguientes mensajes:









Para el caso especial de las preguntas podemos ejecutar o no sentencias dependiendo de la respuesta escogida. Por ejemplo, si deseamos salir o no del programa se tiene:

```
ans=questdlg('¿Desea salir del programa?','SALIR','Si','No','No');
if strcmp(ans,'No')
    return;
end
clear,clc,close all
```

La función `strcmp` compara dos *strings* y si son iguales retorna el valor 1 (*true*). *Clear* elimina todas los valores de workspace, *clc* limpia la pantalla y *close all* cierra todos los Guide. Nótese que la secuencia `'Si','No','No'` termina en 'No'; con esto se logra que la parte *No* del cuadro de pregunta esté resaltado. Si terminara en 'Si', la parte *Si* del cuadro de pregunta se resaltaría.

Como podemos ver, estos mensajes de diálogo poseen una sola línea de información. La siguiente sentencia muestra como se puede agregar saltos de línea:

```
errordlg({'Hola','Un', 'Saludo'},'Mensaje de error')
```

Otra función interesante es *uigetfile*, la que nos permite abrir un archivo y obtener su nombre y dirección. Ejecuta el siguiente código en el *Command Window*:

```
>> [FileName Path]=uigetfile({'*.m;*.mdl'}, 'Escoger')
```

Esto presentará la siguiente interfaz:

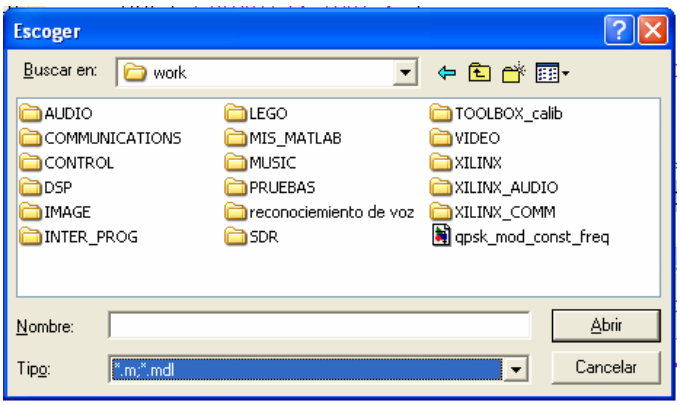


Fig.23. Resultado de la función uigetfile.

Al escoger un programa cualquiera, esta función retorna:

```
FileName =
qpsk_mod_const_freq.mdl
Path =
C:\MATLAB71\work\
```

Si escogemos *cancelar*, esta función retorna:

```
FileName =
     0
Path =
     0
```

Con esta información podemos abrir cualquier archivo o ejecutar cualquier programa, por ejemplo cargar una imagen en un *axes*, abrir un documento Excel o correr un ejecutable.

Creemos una GUI con tres *push-buttons* (el campo *tag* es igual al campo *string*) y un campo *axes* como muestra la siguiente figura:

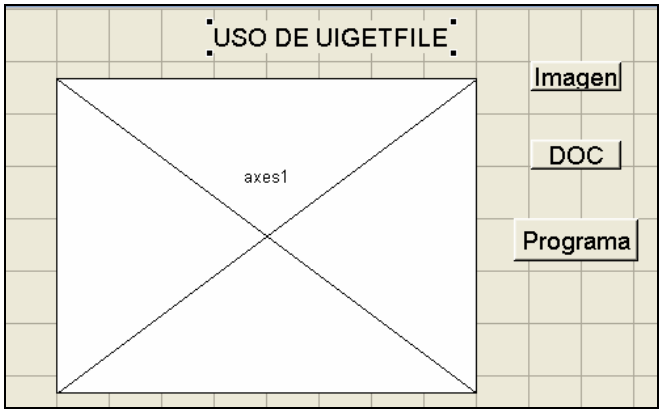


Fig.24. Programa para el uso de uigetfile en una GUI.

Ubicados en la función del botón *imagen*, tenemos el siguiente código:

```
function imagen_Callback(hObject, eventdata, handles)
[FileName Path]=uigetfile({'*.jpg;*.bmp'},'Abrir Imagen');
if isequal(FileName,0)
   return
else
   a=imread(strcat(Path,FileName));
   imshow(a);
end
handles.direccion=strcat(Path,FileName);
guidata(hObject,handles)
```

Como se puede observar, si el usuario presiona *cancelar* el programa no ejecuta ningún proceso. La función *imread* lee una imagen en Matlab e *imshow* la presenta. Este programa tendrá la siguiente presentación:

Fig.25. Uso de uigetfile en una GUI.

El programa del botón *doc* es:

```
function doc_Callback(hObject, eventdata, handles)
[FileName Path]=uigetfile({'*.doc;*.xls'},'Abrir documento');
if isequal(FileName,0)
   return
else
   winopen(strcat(Path,FileName));
end
```

La función *winopen* abre el documento en la dirección especificada por el *path* y el *filename* del archivo.

El programa del botón *programa* es:

```
function programa_Callback(hObject, eventdata, handles)
[FileName Path]=uigetfile({'*.exe'},'Abrir documento');
if isequal(FileName,0)
   return
else
   winopen(strcat(Path,FileName));
end
```

PROGRAMA *FUNC_TRIG*

Con este ejemplo se pretende mostrar el uso del componente *Axes* así como de *pop-up menu* en un GUI que grafica la función seno, coseno y la suma del coseno y el seno.

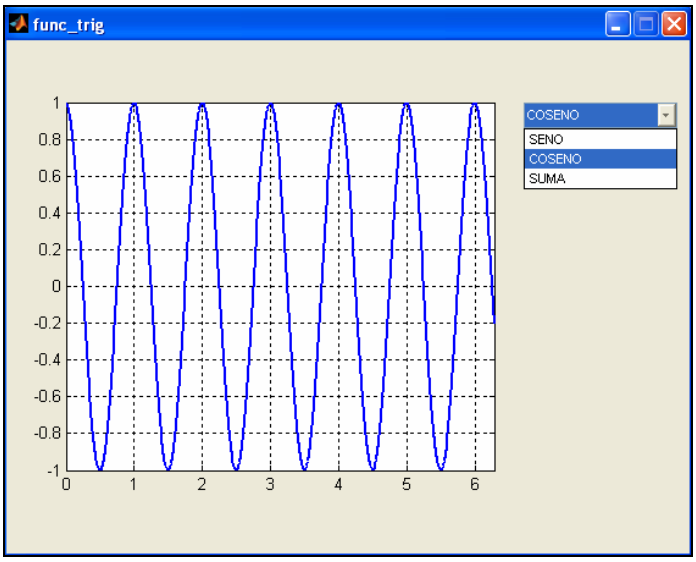


Fig. 26. Entorno del ejemplo func_trig.

Creamos un nuevo GUI con el nombre *func_trig*. Añadimos al mismo un componente *axes* y un *pop-up menu*. Editamos el *string* del *pop-up menú* con *seno* (↵), *coseno* (↵) y *suma*.

Todo m-file posee un campo donde se puede programar las *condiciones iniciales* que tendrá nuestro diseño. Por defecto, se etiqueta `<nombre_de_nuestro diseño>_OpeningFcn`. En nuestro ejemplo, está etiquetada como `func_trig_OpeningFcn`. Debajo de esta línea, editamos el siguiente código:

```
handles.ejex=0:pi/360:2*pi;
y1=sin(2*pi*1*handles.ejex);
plot(handles.ejex,y1,'LineWidth',2);grid on;
axis([0 2*pi -1 1]);
```

Con un click derecho en pop-up menú, nos ubicamos en `View Callbacks>>Callbacks`. Esto nos lleva a la subrutina `function popupmenu1_Callback(hObject, eventdata, handles)`, donde editaremos las siguientes líneas de código:

```
fun =get(handles.sel,'Value');
 switch fun
    case     1
      y1=sin(2*pi*1*handles.ejex);
      plot(handles.ejex,y1,'LineWidth',2);grid on;axis([0 2*pi -1 1]);
    case     2
      y2=cos(2*pi*1*handles.ejex);
      plot(handles.ejex,y2,'LineWidth',2);grid on;axis([0 2*pi -1 1]);
    case     3
      y3=sin(2*pi*1*handles.ejex)+cos(2*pi*1*handles.ejex);
```

```
      plot(handles.ejex,y3,'LineWidth',2);grid on;
      axis([0 2*pi min(y3) max(y3) ]);
end
guidata(hObject, handles);
```

El comando `switch fun` determina cual función será graficada.

Es posible reemplazar la sentencia `switch` por un arreglo `if`, como lo muestra el siguiente código:

```
    if fun==1
      y1=sin(2*pi*1*handles.ejex);
      plot(handles.ejex,y1,'LineWidth',2);grid on;axis([0 2*pi -1 1]);
    elseif   fun==2
      y2=cos(2*pi*1*handles.ejex);
      plot(handles.ejex,y2,'LineWidth',2);grid on;axis([0 2*pi -1 1]);
    else
      y3=sin(2*pi*1*handles.ejex)+cos(2*pi*1*handles.ejex);
      plot(handles.ejex,y3,'LineWidth',2);grid on;
      axis([0 2*pi min(y3) max(y3) ]);
end
guidata(hObject, handles);
```

Y nuestro programa funciona de igual manera. Esto es todo en cuanto a este ejemplo. Suerte en su diseño.

PROGRAMA *LISTBOX*

Cos este ejemplo, mostraré el uso de *listbox*. Creamos un nuevo GUI etiquetado como *Listbox.* Añadimos al diseño un *listbox* y un par de *statictext* y los ordenamos como lo muestra la figura.

Fig. 27. Entorno del ejemplo Listbox.

Haciendo doble-click en *listbox* editamos los elementos de la lista. Puedes colocar los nombres que desees.

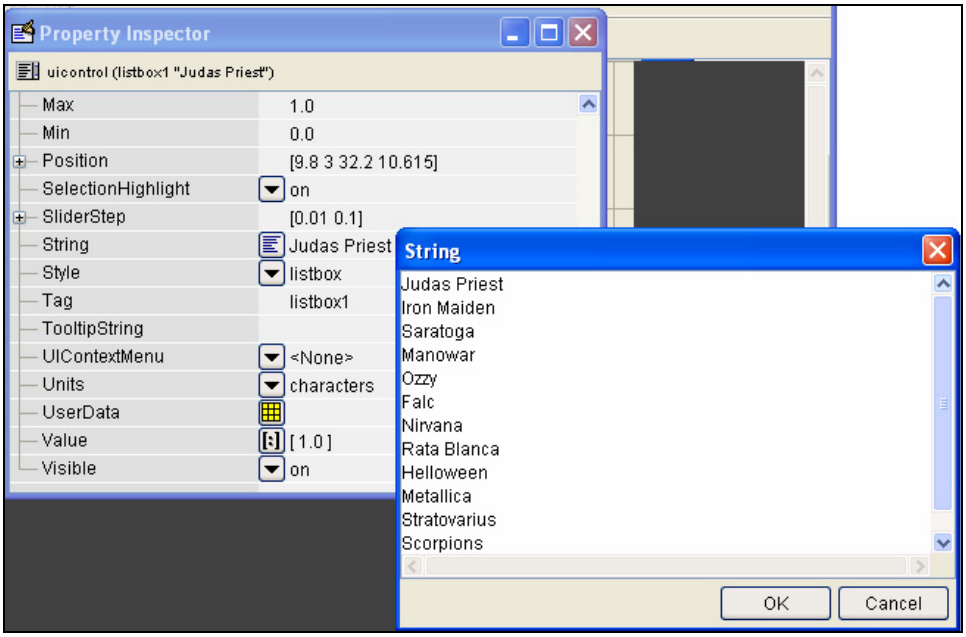


Fig. 28. Edición de los elementos de listbox.

Con un click derecho en *listbox*, nos ubicamos en *View Callbacks>>Callbacks.* Esto nos lleva a la m-file donde se encuentra la subrutina del *listbox*.

Editamos el siguiente código:

```
inf=get(hObject,'Value');
gos=get(hObject,'String');

switch inf
    case 1
        set(handles.txt1,'string',...
['Grupo británico. Inicio su carrera en 1973.',...
 ' Último album: *Angel of Retribution (2005)*.']);
    case 2
        set(handles.txt1,'string',...
['Grupo británico. Inicio su carrera en 1979.',...
 ' Último album: *Dance of death (2005)*.']);
    case 3
        set(handles.txt1,'string',...
['Grupo español. Inicio su carrera en 1999.',...
 ' Último album: *Agotarás (2004)*.']);
    case 4
        set(handles.txt1,'string',...
['Grupo estadounidense. Inicio su carrera en 1990.',...
 ' Último album: *Louder than hell (2005)*.']);
    case 5
        set(handles.txt1,'string',...
['Solista británico. Inicio su carrera en 1971.',...
 ' Último album: *Ozzfest (2006)*.']);
    case 6
        set(handles.txt1,'string',...
['Grupo ecuatoriano. Inicio su carrera en 1995.',...
 ' Último album: *Prisionero del tiempo (2005)*.']);
    case 7
        set(handles.txt1,'string',...
['Grupo estadounidense. Inicio su carrera en 1995.',...
 ' Último album: *Prisionero del tiempo (2005)*.']);
    case 8
        set(handles.txt1,'string',...
['Grupo argentino. Inicio su carrera en 1989.',...
 ' Último album: *El camino del fuego (1994)*.']);
    case 9
        set(handles.txt1,'string',...
['Grupo alemán. Inicio su carrera en 1983.',...
 ' Último album: *Keeper OSK III (2005)*.']);
    case 10
        set(handles.txt1,'string',...
['Grupo estadounidense. Inicio su carrera en 1983.',...
 ' Primer album: *Kill them all (1983)*.']);
    case 11
        set(handles.txt1,'string',...
['Grupo finlandés. Inicio su carrera en 1984.',...
 ' Último album: *Infinity (2005)*.']);
    case 12
        set(handles.txt1,'string',...
['Grupo alemán. Inicio su carrera en 1973.',...
 ' Último album: *Eye by eye (2005)*.']);
end

guidata(hObject,handles);
```

Recuérdese que podemos cambiar la sentencia `switch` por un `if-esleif-else`.

En la parte de inicialización del programa `function Listbox_OpeningFcn(hObject, eventdata, handles, varargin)` editamos las siguientes sentencias:

```
set(handles.text2,'string','List Box');
set(handles.txt1,'string',...
['Grupo británico. Inicio su carrera en 1973.',...
 ' Último álbum: *Angel of Retribution (2005)*.']);
```

En el siguiente ejemplo, asignaremos a una lista 2 la posición de la lista 1. Se hará simplemente una conversión de potencia en vatios a una potencia en decibeles.

El campo *tag* del primer *list box* es *lista1* y del segundo *lista2*. Las condiciones iniciales son:

```
function list_box_2_OpeningFcn(hObject, eventdata, handles, varargin)

set(handles.lista1,'String',[10 20 30 40 50 60]);
set(handles.lista2,'String',[db(10,'Power') db(20,'Power')...
    db(30,'Power') db(40,'Power') db(50,'Power') db(60,'Power')]);
```

El callback de la primera lista es:

```
function lista1_Callback(hObject, eventdata, handles)
a=get(hObject,'Value');
set(handles.lista2,'Value',a);
```

La siguiente figura muestra como funciona el programa:

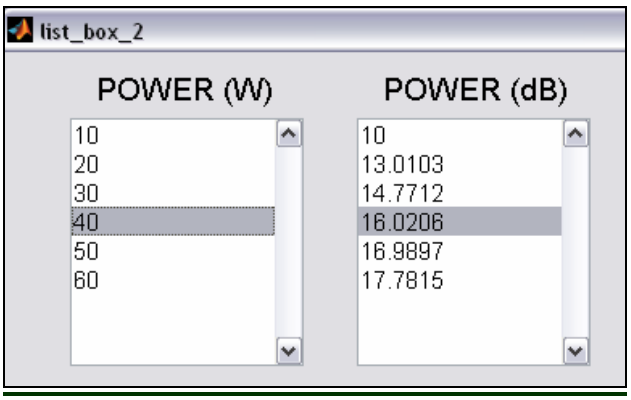


Fig. 29. Listas concurrentes.

PROGRAMA *SLIDER*

Con este ejemplo pretendemos mostrar el uso del *slider*, así como de un *check box*. En sí, este programa es un detector de nivel de datos ingresados por el usuario.

Fig. 30. Entorno del programa Slider.

Iniciamos un nuevo GUI con el nombre *Slider*. Añadimos los componentes que muestra la figura anterior. Editamos el *checkbox* como muestra la siguiente figura:

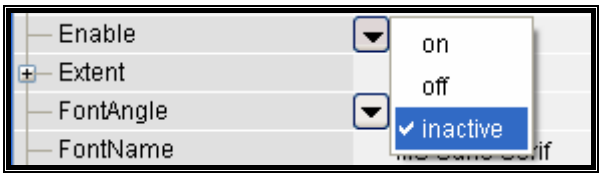


Fig. 31. Habilitación opción inactive.

Enable en estado *inactive* permite escribir sobre el *checkbox* y no poder modificar este estado por el usuario.

La función del *slider* queda de la siguiente manera:

```
function slider1_Callback(hObject, eventdata, handles)
handles.slider1=get(hObject,'Value'); %Carga en handles.slider1
el valor delSlider
handles.slider1=100.*handles.slider1;
set(handles.text1,'String',handles.slider1); %Escribe el valor de
Slider en statictext
if (handles.slider1 < 50)
    set(handles.text3,'String','BUEN Nivel');
    set(handles.checkbox1,'Value',0) ;
else
    set(handles.text3,'String','MAL Nivel');
```

```
    set(handles.checkbox1,'Value',1);
    %beep
end
guidata(hObject, handles);
```

El conjunto de la secuencia `if` determina dos estados: 1) Si el valor del *slider* es menor que 50 escribe en el componente *text3* el texto "*BUEN Nivel*" y mantiene en cero el estado del *checkbox*; 2) Si el valor del *slider* es mayor o igual que 50 escribe en el componente *text3* el texto "MAL Nivel" y mantiene en uno el estado del *checkbox*.

Como en un ejemplo anterior, la imagen de fondo la añadimos con las sentencias:

```
[x,map]=imread('Stratovarius.jpg','jpg');
image(x),colormap(map),axis off,hold on
```

Realicemos un programa más interesante con el uso del slider: cambiar el contraste de una imagen. En una nueva GUI colocamos un *slider*, un *push-button* y un *axes* como muestra la figura:

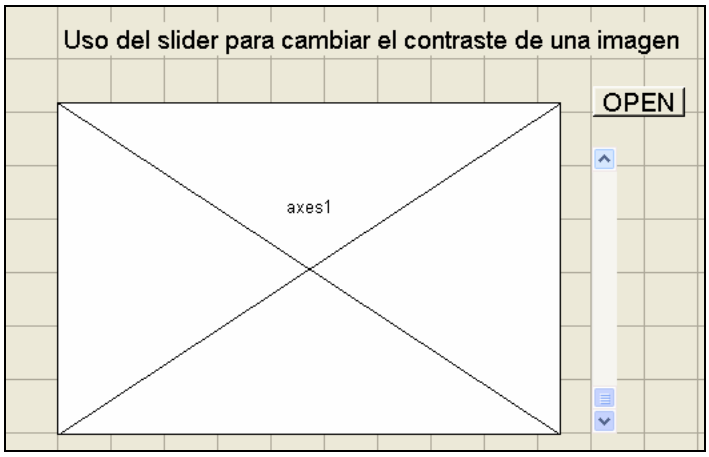


Fig.32. Contraste de una imagen con el empleo del slider.

El campo tag del botón es *open*, igual que el *string*. La programación es la misma que se usó para abrir una imagen:

```
function open_Callback(hObject, eventdata, handles)
[FileName Path]=uigetfile('*.jpg','Abrir Imagen');
if isequal(FileName,0)
   return
else
   a=imread(strcat(Path,FileName));
   imshow(a);
end
handles.direccion=strcat(Path,FileName);
guidata(hObject,handles)
```

El campo *tag* del *slider* es el mismo que aparece por defecto. La programación del *slider* es:

```
function slider1_Callback(hObject, eventdata, handles)
a=0.02*get(hObject,'Value');
```

```
if handles.direccion==0
    return
else
b=rgb2gray(imread(handles.direccion));
c=double(b); d=a*c; imshow(d);
end
```

La función *rgb2gray* transforma una imagen a color a escala de grises. La función *double* cambia el formato entero sin signo de 8 bits (*uint8*) de la imagen a un formato decimal, de tal forma que sea posible multiplicar la imagen por los valores de slider.

Para evitar errores al mover el *slider* cuando no se ha cargado ninguna imagen aún, en la parte de las condiciones iniciales del programa se inicializa el valor del identificador *handles.direccion* con el valor de 0:

```
function slider_contraste_OpeningFcn(hObject, eventdata, handles,
varargin)
handles.direccion=0;
% Choose default command line output for slider_contraste
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
```

El programa en funcionamiento se muestra en las siguientes figuras:

Fig.33. Cambio del contraste de una imagen.

Para que la escala del slider contenga valores positivos y negativos, simplemente se edita como muestra la siguiente figura:

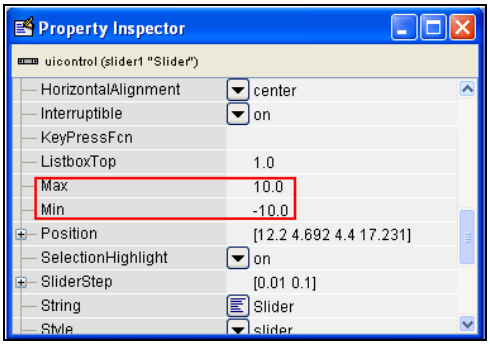


Fig. 34. Escala entera del slider.

BOTONES PERSONALIZADOS

Fig. 35. Personalización de botones.

La interfaz gráfica de usuario de MATLAB nos permite personalizar la presentación de nuestros botones como lo muestra la figura precedente. Para lograr esto, editamos el siguiente código en la parte del m-file destinada a la inicialización del programa (`function Etiqbutton_OpeningFcn…`):

```
%Carga la imagen de fondo (opcional)
[x,map]=imread('hammerfall.jpg','jpg');
image(x),colormap(map),axis off,hold on

%Coloca una imagen en cada botón
[a,map]=imread('vol.jpg');
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/30);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.pushbutton1,'CData',g);

[a,map]=imread('stop.jpg');
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/30);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.pushbutton2,'CData',g);

[a,map]=imread('play.jpg');
[r,c,d]=size(a);
x=ceil(r/30);
```

```
y=ceil(c/30);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.pushbutton3,'CData',g);

[a,map]=imread('open_files.jpg');
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/30);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.pushbutton4,'CData',g);

[a,map]=imread('cd_eject.jpg');
[r,c,d]=size(a);
x=ceil(r/35);
y=ceil(c/35);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.pushbutton5,'CData',g);

[a,map]=imread('pause.jpg');
[r,c,d]=size(a);
x=ceil(r/100);
y=ceil(c/80);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.pushbutton6,'CData',g);

[a,map]=imread('mute2.jpg');
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/30);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.pushbutton7,'CData',g);
% Choose default command line output for Etiqbutton
handles.output = hObject;
```

Para personalizar aún más, se puede hacer aparecer el nombre del botón cuando se acerca el cursor, simplemente llenando el campo *TooltipString* en el *Property Inspector*.

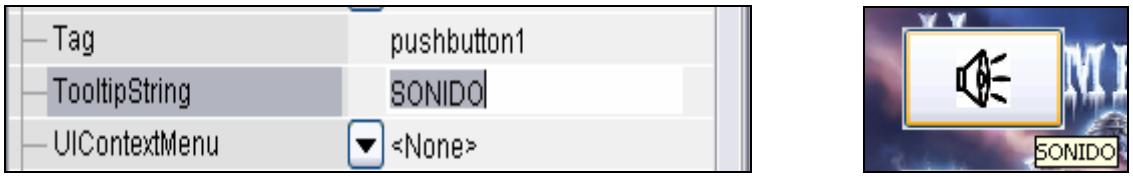


Fig. 36. Campo TooltipString para etiquetar botón.

PROGRAMA *IMAGENES*

Con este sencillo programa se mostrará como asignar a cada *axes* de un Guide un gráfico o imagen específico.

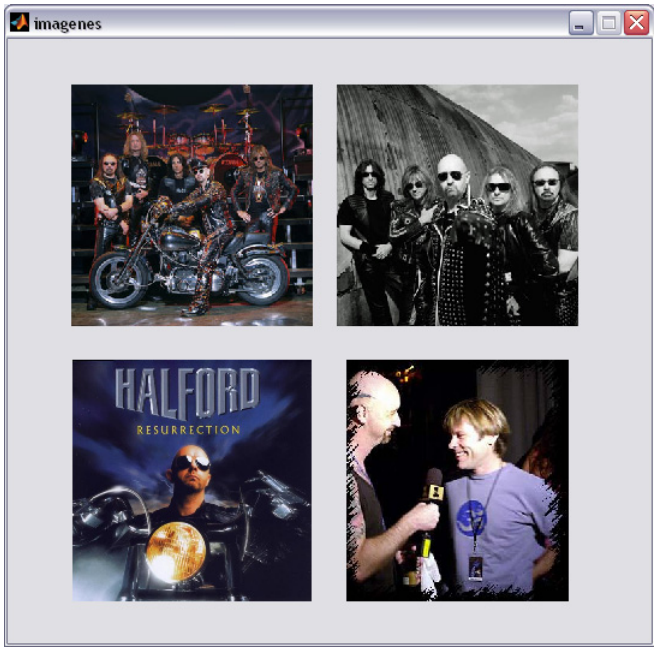


Fig. 37. Programa Imágenes.

Ingrese en un nuevo guide los cuatro elementos que se muestran en la figura siguiente:

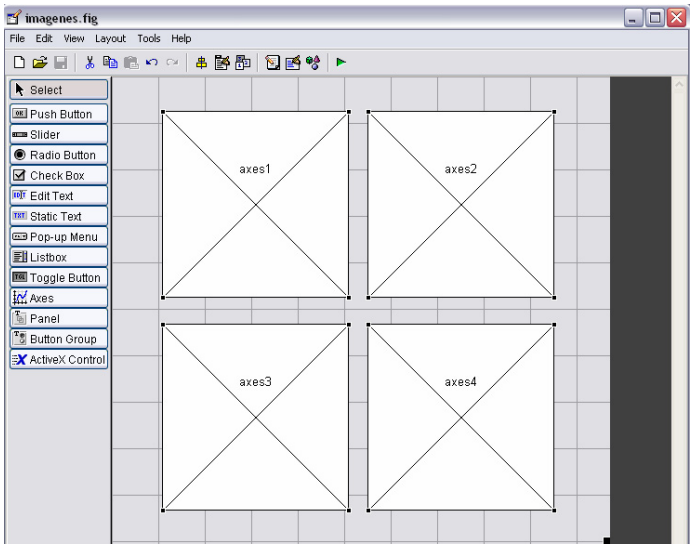


Fig. 38. Programa Imágenes.

En la parte de inicialización del programa editamos lo siguiente:

```
axes(handles.axes1)
background = imread('una.jpg');
axis off;
imshow(background);
%*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

```
axes(handles.axes2)
background = imread('dos.jpg');
axis off;
imshow(background);
%*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
axes(handles.axes3)
background = imread('tres.jpg');
axis off;
imshow(background);
%*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
axes(handles.axes4)
background = imread('cuatro.jpg');
axis off;
imshow(background);
```

Los nombres de cada imagen son *una*, *dos*, *tres* y *cuatro*. Recuerde que las imágenes deben estar en la misma carpeta que está el programa.

PROGRAMA *ELEMENTOS*

Con este programa pretendemos mostrar el uso de *toggle button*, así como el empleo de *panel button*.

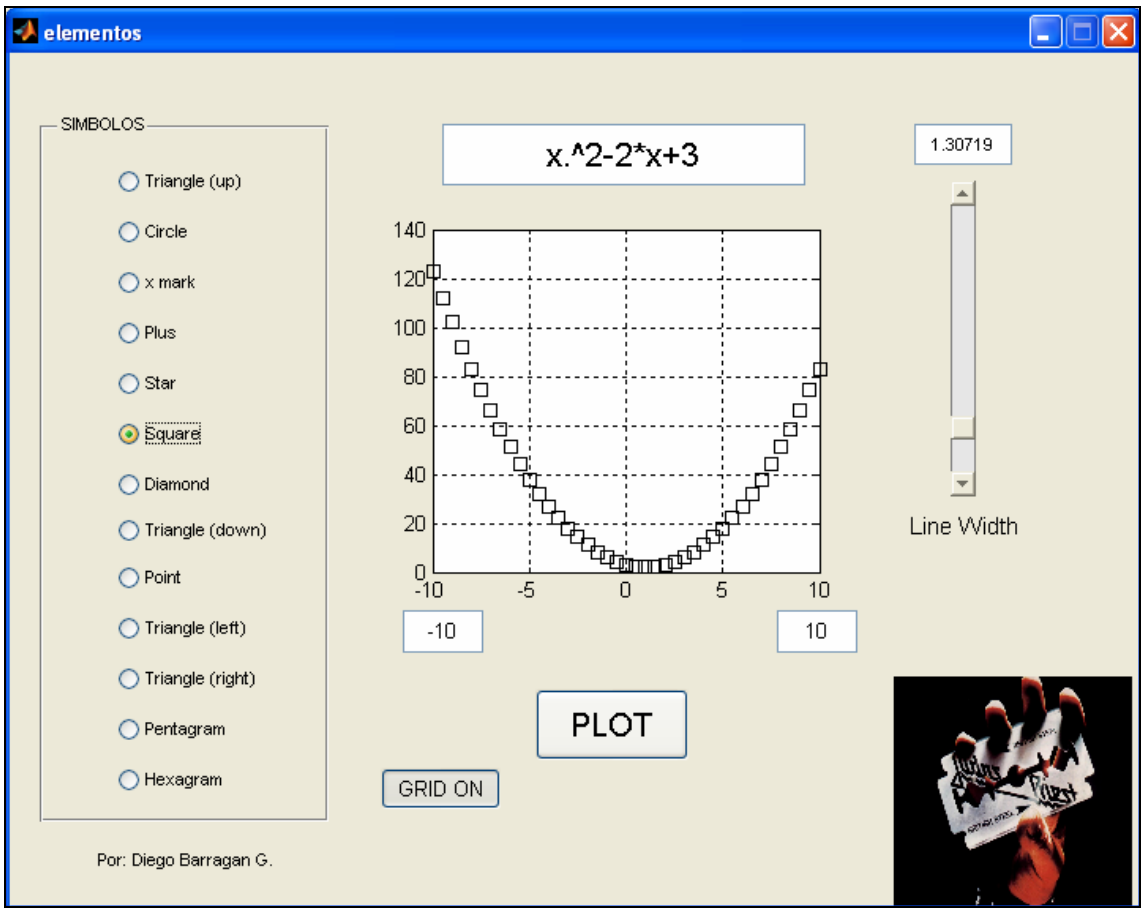


Fig. 39. Entorno del programa Elementos.

Una de las novedades que presenta la versión 7.0.1 de MATLAB en la interfaz gráfica de usuario es el *panel de botones (button panel)*. En este ejemplo, los botones del panel deben tener la condición de que solo uno de ellos deber estar en alto y los demás en estado bajo, para así poder configurar el estilo de línea. Además, el botón encendido debe permanecer es estado alto aunque se lo vuelva a seleccionar.

La configuración de los parámetros de la función a graficar se basa en la siguiente tabla:

Color	Símbolo	Tipo línea
b blue	. point	- solid
g green	o circle	: dotted
r red	x x-mark	-. dashdot
c cyan	+ plus	-- dashed
m magenta	* star	
y yellow	s square	
k black	d diamond	
	v triangle (down)	
	^ triangle (up)	
	< triangle (left)	
	> triangle (right)	
	p pentagram	
	h hexagram	

El sector del m-file destinado a la programación de las condiciones iniciales tiene el siguiente código:

```
% Asignación del objeto axes para la imagen adjunta.
axes(handles.imagen)
[x,map]=imread('britishsteel.jpg','jpg');
image(x),colormap(map),axis off,hold on

% Asignación del campo axes para la función a graficar.
axes(handles.grafica)
x=-10:0.5:10;           %Primer intervalo de graficación.
handles.x=x;
handles.h=plot(x,x.^2); %Graficar una parábola.
set(handles.h,'LineStyle','^','Color',[0.5 0 0]);%Estilo de línea
grid off;                                        %Grid inactivo
set(handles.grids,'String','GRID OFF');%Etiquetar el toggle button.
title ('SEÑAL');
handles.slider1=0.1; %Inicializar el valor del slider.
set(handles.edit1,'String',handles.slider1);%Mostrar el valor del
slider.
```

Una vez que se ha colocado cada uno de los botones dentro del *panel de botones* y se los ha etiquetado, se ejecuta lo que muestra la siguiente figura:

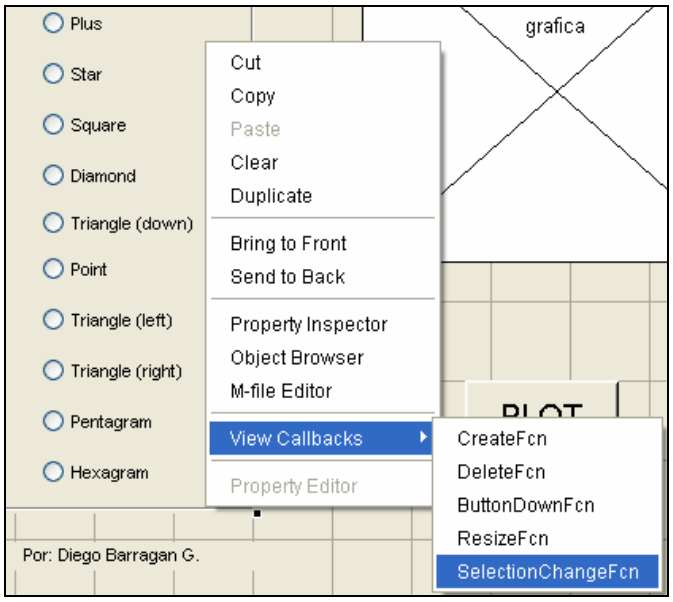


Fig. 40. Crear una función de Selección.

En el m-file asociado se creará el siguiente código:

```
function uipanel1_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to uipanel1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Para este ejemplo, el campo *Tag* de de cada botón del panel se ha etiquetado *uno, dos, tres…* De esta forma, es más fácil trabajar cada configuración en el m-file. De esta forma, la programación del panel de botones queda de la siguiente manera:

```
function uipanel1_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to uipanel1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if (hObject==handles.uno)
    set(handles.h,'LineStyle','.','Color','g');
elseif(hObject==handles.dos)
    set(handles.h,'LineStyle','o','Color','r');
elseif(hObject==handles.tres)
    set(handles.h,'LineStyle','x','Color','c');
elseif(hObject==handles.cuatro)
    set(handles.h,'LineStyle','+','Color','m');
elseif(hObject==handles.cinco)
    set(handles.h,'LineStyle','*','Color','y');
elseif(hObject==handles.seis)
    set(handles.h,'LineStyle','s','Color','k');
elseif(hObject==handles.siete)
    set(handles.h,'LineStyle','d','Color',[1 0.5 0.5]);
elseif(hObject==handles.ocho)
    set(handles.h,'LineStyle','v','Color',[1 0.5 1]);
elseif(hObject==handles.nueve)
    set(handles.h,'LineStyle','^','Color',[0.5 0 0]);
elseif(hObject==handles.diez)
    set(handles.h,'LineStyle','>','Color',[0.5 0 1]);
elseif(hObject==handles.once)
    set(handles.h,'LineStyle','<','Color',[0.5 1 0]);
```

```
elseif(hObject==handles.doce)
    set(handles.h,'LineStyle','p','Color',[0.5 1 1]);
else
    set(handles.h,'LineStyle','h','Color',[0.52 0 0]);
end
```

La función del *slider* queda con el siguiente código:

```
function slider1_Callback(hObject, eventdata, handles)
handles.slider1=get(hObject,'Value');
handles.slider1= handles.slider1*10;
if handles.slider1==0
    handles.slider1=handles.slider1+0.01;
end
set(handles.h,'LineWidth',handles.slider1);
set(handles.edit1,'String',handles.slider1);
guidata(hObject,handles);
```

El valor por defecto del *slider* va de 0 a 1. Por esta razón se lo multiplica por 10. Como el valor capturado del *slider* se usa para el ancho de línea de la gráfica, este valor no deber ser cero y es por eso que usamos una sentencia *if*, para que cuando el valor ingresado sea cero, inmediatamente se le sume 0.01. La sentencia `set(handles.h,'LineWidth',handles.slider1)` modifica el valor del ancho de línea de la señal y la sentencia `set(handles.edit1,'String',handles.slider1)` escribe el valor numérico en *edit1*.

El campo donde se ingresa la ecuación tiene el siguiente código:

```
function edit2_Callback(hObject, eventdata, handles)
ini=get(hObject,'String');
handles.edit2=ini;
guidata(hObject,handles);
```

La configuración del *toggle button* (etiquetado como *grids*) que añade el grid, tiene el código siuiente:

```
function grids_Callback(hObject, eventdata, handles)
die=get(hObject,'Value');
handles.die=die;
if handles.die==1
    grid on;
    set(handles.grids,'String','GRID ON');
else
    grid off;
    set(handles.grids,'String','GRID OFF');
end
```

La sentencia `set(handles.grids,'String','GRID ON')` modifica el texto del *toggle button*, de esta forma se leerá GRID ON o GRID OFF dependiendo de la selección del botón.

Al ingresar la función a graficar, el botón PLOT ejecutará el siguiente código:

```
function plot_Callback(hObject, eventdata, handles)
axes(handles.grafica)                    %Grafica en 'grafica'
```

```
%men y may almacena el valor del intervalo de graficación.
men=get(handles.menor,'String');
may=get(handles.mayor,'String');

%Estos valores se transforman a formato double y se alamacenan en x.
x=str2double(men):0.5:str2double(may);  %intervalo de gráfica

%Salvamos el valor ingresado de la función y pasa a formato String.
ini=handles.edit2;                      %Tomar la Ecuación
ini=char(ini);                          %Pasar a formato char

%Salvar los valores acutales de graficación.
jud=get(handles.h,'Marker');          %TOmar el valor LineStyle actual
asp=get(handles.h,'LineWidth');
rie=get(handles.h,'Color');
handles.h=plot(x,eval(ini));          %Graficar la ecuación ingresada
set(handles.h,'LineStyle',jud);       %Mantener el valor LineStyle
set(handles.h,'LineWidth',asp);
set(handles.h,'Color',rie);

%Tomar el valor de GRID y mantenerlo en la nueva gráfica.
handles.die=get(handles.grids,'Value');
if handles.die==1
    grid on;
    set(handles.grids,'String','GRID ON');
else
    grid off;
    set(handles.grids,'String','GRID OFF');
end

guidata(hObject,handles)
```

PROGRAMA GUIDE_SIMULINK

Con este programa se pretende mostrar la interacción entre una Interfaz Gráfica y Simulink.

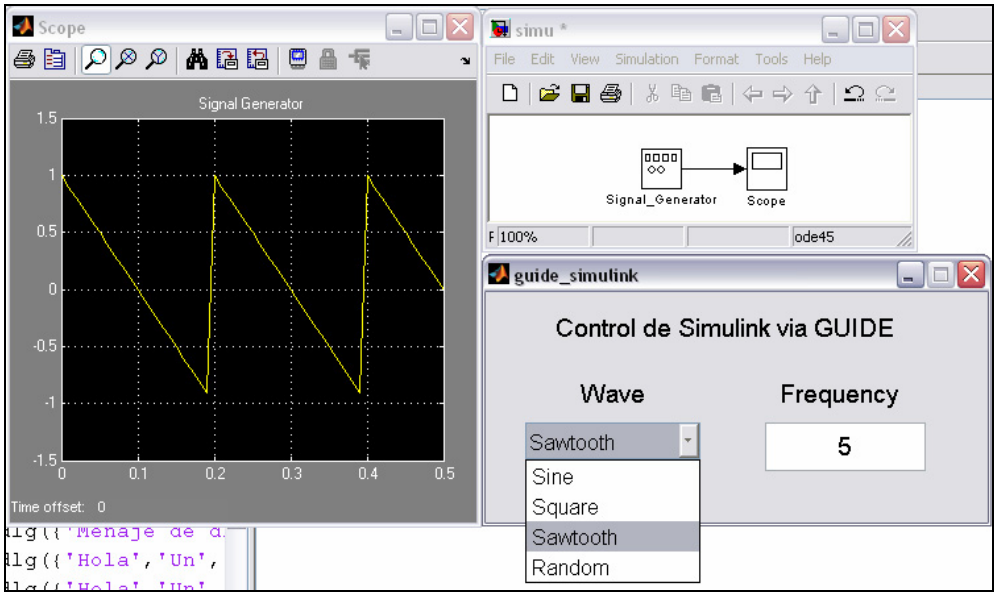


Fig. 41. Entorno programa GUIDE_SIMULINK

Este sencillo programa controla dos parámetros del bloque *Signal_Generator*: la forma de onda y la frecuencia.

En la parte de inicialización del programa (`guide_simulink_OpeningFcn`) se escribe lo siguiente:

```
find_system('Name','simu');
open_system('simu');
set_param('simu/Signal_Generator','Wave','sine');
set_param('simu/Signal_Generator','frequency','5');
set_param(gcs,'SimulationCommand','Start');
```

Estos comandos son los primeros que se ejecutan al abrir la interfaz gráfica. *Find_system* y *open_system* son para comprobar si existe el programa en simulink y para abrirlo. La sentencia para escribir en los bloques de simulink es *set_param*, que se usa para establecer en el parámetro *Wave form* del bloque *Signal_Generator* la onda *sine*. La sintaxis de *set_param* es:

```
set_param('nombre_del_programa/nombre_del_bloque','parámetro','valor')
```

El comando `set_param(gcs,'SimulationCommand','Start')` es para iniciar la ejecución del programa en simulink.

La programación del *Pop-up menú* (etiquetada el en campo *Tag* como *wave*) es como sigue:

```
function wave_Callback(hObject, eventdata, handles)
onda = get(hObject,'Value');
if onda==1
```

```
    set_param('simu/Signal_Generator','Wave','sine');
    set_param(gcs,'SimulationCommand','Start');
elseif onda==2
    set_param('simu/Signal_Generator','Wave','square');
    set_param(gcs,'SimulationCommand','Start');
elseif onda==3
    set_param('simu/Signal_Generator','Wave','sawtooth');
    set_param(gcs,'SimulationCommand','Start');
else
    set_param('simu/Signal_Generator','Wave','random');
    set_param(gcs,'SimulationCommand','Start');
end
```

La programación del *edit text* que manipula la frecuencia es:

```
function edit1_Callback(hObject, eventdata, handles)
fre = get(hObject,'String');
set_param('simu/Signal_Generator','frequency',fre);
set_param(gcs,'SimulationCommand','Start');
```

Otro programa sencillo es el siguiente:

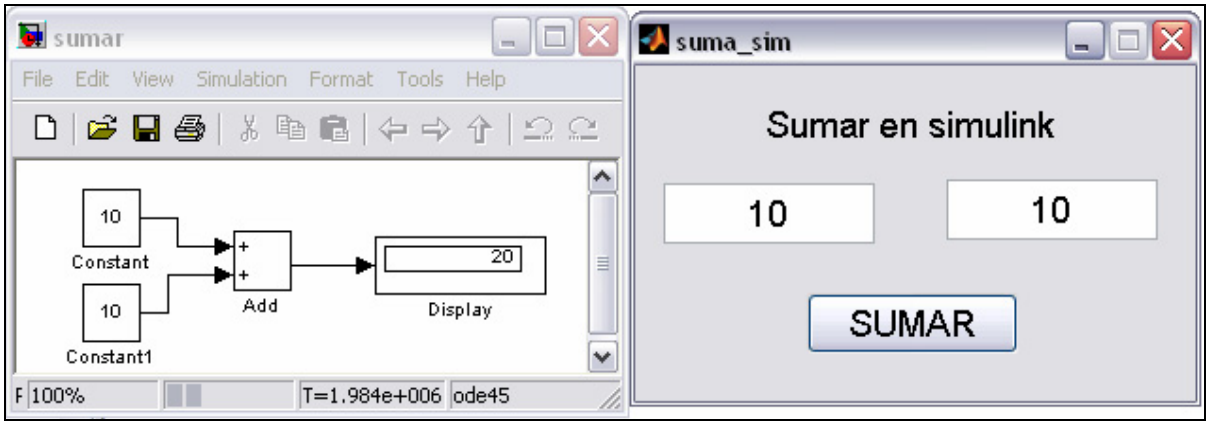


Fig. 42. Programa suma_sim.

Como se ve en la figura precedente, los sumandos se ingresan desde un Guide y son evaluados en simulink.

La parte de condiciones iniciales es como sigue:

```
find_system('Name','sumar');
open_system('sumar');
set_param(gcs,'SimulationCommand','Start');
```

Y la parte del pushbutton es:

```
val_a=get(handles.valor_a,'String');
val_b=get(handles.valor_b,'String');
set_param('sumar/Constant','Value',val_a);
set_param('sumar/Constant1','Value',val_b);
```

Para finalizar con los programas de interacción entre GUIDE y Simulink, se realizará un simple reproductor de audio, en el cual se podrá ejecutar *play*, *pausa*,

continuar y *stop*; como también tener control del volumen de la reproducción. La figura siguiente muestra el entorno de este sencillo programa:

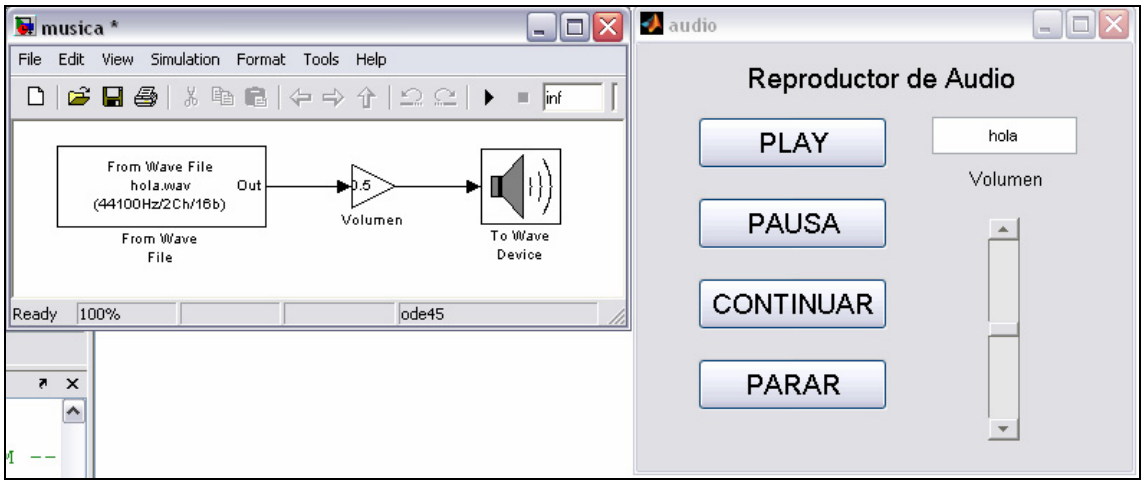


Fig. 43. Entorno del programa audio.

Los componentes del programa *musica* están en la librería *Signal Processing Blockset*. Es necesario mantener la etiqueta de cada bloque de simulink.

Las condiciones iniciales del GUIDE audio es como sigue:

```
1. set(handles.volumen,'Value',0.5);
2. find_system('Name','musica');
3. open_system('musica');
4. set_param('musica/Volumen','Gain','0.5');
5. musica=get(handles.cancion,'String');
6. set_param('musica/From Wave File','FileName',char(musica));
```

La línea 1 establece el valor de 0.5 al *slider*, que al abrir el programa estará justo en la mitad. Las líneas 2 y 3 son para encontrar el programa y abrirlo. La línea 4 establece en 0.5 el valor del parámetro *Gain* del bloque *Volumen* en Simulink. La línea 5 toma el nombre del archivo de audio (*.wav)[3] que se reproducirá (recuérdese que el archivo de audio debe estar en la misma carpeta del programa) y la línea 6 asigna este nombre al bloque *From Wave File*.

El fin de este programa, más allá de reproducir audio, es controlar la ejecución, pausa, continuación y parada de un programa en simulink desde un Guide.

El campo *tag* de los botones *play*, *pausa*, *continuar* y *parar* es *play*, *pause*, *contin* y *stopp* respectivamente. La programación de cada botón es como sigue:

```
function play_Callback(hObject, eventdata, handles)
musica=get(handles.cancion,'String');
set_param('musica/From Wave File','FileName',char(musica));
set_param(gcs,'SimulationCommand','Start');

function pause_Callback(hObject, eventdata, handles)
set_param(gcs,'SimulationCommand','Pause')
```

[3] Para convertir archivos de audio a formato *wav* he utilizado el programa *MUSICMATCH*, en la parte de *Archivo→Convertir Archivos…*

```
function stopp_Callback(hObject, eventdata, handles)
set_param(gcs,'SimulationCommand','Stop')

function contin_Callback(hObject, eventdata, handles)
set_param(gcs,'SimulationCommand','Continue')
```

Para el control del volumen se ha escogido un valor mínimo de 0 y un máximo de 1, que son los valores por defecto del *slider*. El *slider* se ha etiquetado (*tag*) como *volumen*. La programación del volumen queda como sigue:

```
function volumen_Callback(hObject, eventdata, handles)
volu=get(hObject,'Value');
set_param('musica/Volumen','Gain',num2str(volu));
```

NOTA: es importante no exagerar en los valores del parámetro *Gain* del bloque *Volumen* en simulink, ya que es muy posible que vuestros parlantes o audífonos sufran daño innecesario.

PROGRAMA *SECUENCIA*

Con este programa se pretende mostrar la forma de intercambiar información entre dos GUI distintas. Por ejemplo, en un programa de Chat entre dos PCs a través de puerto serial, los valores de baudios, puerto, bits de datos, etc., se pueden ingresar en una GUI aparte, mejorando la presentación del programa.

Para intercambiar datos entre dos GUI simplemente declaramos los datos como variables globales. Ejemplo: `global dato.`

Para este ejemplo, en una GUI se ingresan dos valores que serán sumados en otra GUI.

La GUI de ingreso de datos la he llamado *secuencia*.

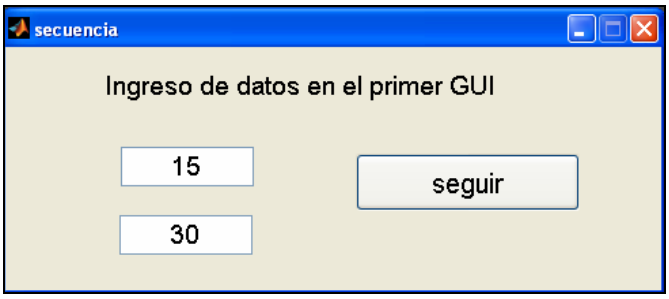


Fig. 44. Entorno del programa secuencia.

Los campos *Tag* de cada *edit text* son *uno* y *dos*. La programación del botón *seguir* (Tag: SIGUIENTE) es como sigue:

```
function SIGUIENTE_Callback(hObject, eventdata, handles)
%Se declaran las variables como globales

global un
global do
un=str2double(get(handles.uno,'String'));
do=str2double(get(handles.dos,'String'));
close secuencia %Cierra el GUI actual
seguir          %Abre el siguiente GUI llamado seguir
```

El programa donde se suma estos valores se llama *seguir*.

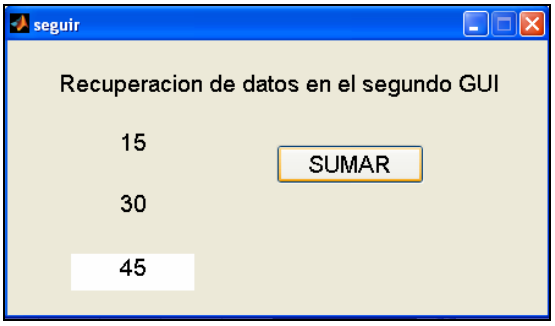


Fig. 45. Entorno del programa seguir.

En la parte del M-file donde se programan las condiciones iniciales del GUI editamos lo siguiente:

```
function seguir_OpeningFcn(hObject, eventdata, handles, varargin)
%Nuevamente se declaran las variables como globales, con el mismo
%nombre
%Las asignamos a los statictext text1 y text2
global un
global do
set(handles.text1,'String',un);
set(handles.text2,'String',do);
% Choose default command line output for seguir
handles.output = hObject;
```

Con esto obtenemos los datos ingresados en el otro GUI.

Para sumarlo, el código del botón *sumar* es:

```
function sumar_Callback(hObject, eventdata, handles)
global un
global do
s=un + do;
set(handles.text4,'String',s);
```

PROGRAMA *videoGUI*

Con este programa se pretende mostrar la forma de capturar video e imágenes en una interfaz gráfica. Este programa funciona sobre Matlab 7.1.

Fig. 46. Programa para captura de video e imágenes.

El campo *Tag* del *axes* que presenta el video es *video_cam* y del *axes* que contiene la imagen capturada es *fotografia*. El código en la sección de inicialización del programa es el siguiente:

```
function videoGUI_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
% Update handles structure
handles.rgb = [];
handles.noback = [];
guidata(hObject, handles);

% This sets up the video camera and starts the preview
% Only do this when it's invisible
if strcmp(get(hObject,'Visible'),'off')
  try
    handles.vidobj = videoinput('winvideo');
    % Update handles structure
    start(handles.vidobj);
    guidata(hObject, handles);
    vidRes = get(handles.vidobj, 'VideoResolution');
    nBands = get(handles.vidobj, 'NumberOfBands');
    hImage = image(zeros(vidRes(2), vidRes(1), nBands), 'Parent',...
                   handles.video_cam);
    preview(handles.vidobj,hImage);
  catch
    msgbox('NO HAY CÁMARA CONECTADA. Cargando Profile.jpg.')
    hImage = image(imread('profile.jpg'), 'Parent',handles.video_cam);
  end
end
```

La sentencia `strcmp` compara dos *strings*. Si son iguales retorna 1 y caso contrario, retorna 0. El código entre *try* y *catch* establece la conexión de video e inicia la reproducción. En caso de producirse un error en estas líneas, se ejecuta el código entre *catch* y *end*, que coloca una imagen ('profile.jpg') en el *axes* de captura de foto.

El código para capturar la imagen es:

```
function captura_Callback(hObject, eventdata, handles)
try
    handles.rgb = getsnapshot(handles.vidobj);
catch
    handles.rgb = imread('profile.jpg');
end
% Update handles structure
guidata(hObject, handles);
image(handles.rgb,'Parent',handles.fotografia);
axes(handles.fotografia)
axis off
```

La función `getsnapshot` captura la imagen de la cámara de video. En caso de que la cámara se haya desconectado, se carga la imagen *profile.jpg*. Obtenida la imagen, finalmente se la asigna al *axes* etiquetado como *fotografía* con la función `image`.

El código para guardar la imagene es:

```
function guardar_Callback(hObject, eventdata, handles)
rgb = getimage(handles.fotografia);
if isempty(rgb), return, end

%guardar como archivo
fileTypes = supportedImageTypes; % Función auxiliar.

[f,p] = uiputfile(fileTypes);
if f==0, return, end

fName = fullfile(p,f);
imwrite(rgb,fName);
msgbox(['Imagen guardada en ' fName]);

%Cambio al directorio donde se ha guardado la imagen (prescindible)
%if ~strcmp(p,pwd)
%  cd(p);
%end

function fileTypes = supportedImageTypes
% Función auxiliar: formatos de imágenes.
fileTypes = {'*.jpg','JPEG (*.jpg)';'*.tif','TIFF (*.tif)';...
             '*.bmp','Bitmap (*.bmp)';'*.*','All files (*.*)'};
```

PROGRAMA *parallel_port*

Con este sencillo programa se mostrará la forma de leer y escribir valores en el puerto paralelo (PP) de nuestro PC a través de una interfaz gráfica.

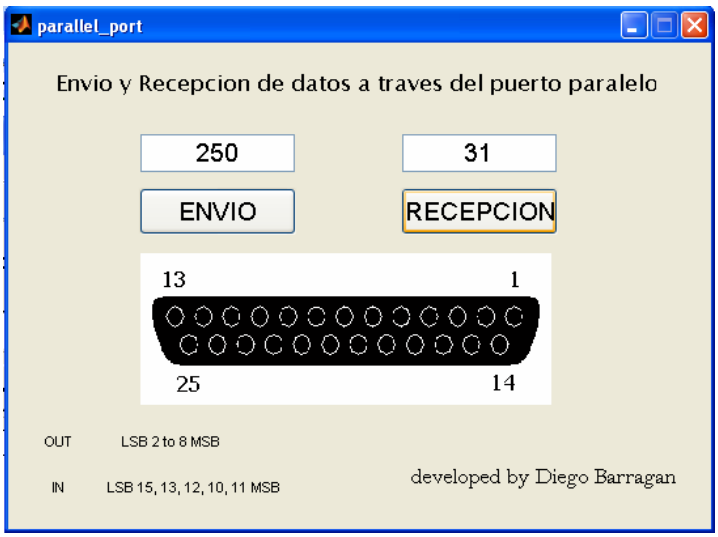


Fig. 47. Programa para leer y escribir el PP.

Antes de empezar con el diseño de la interfaz, es necesario conocer como Matlab trabaja con el puerto paralelo.

En el *command window* ejecutamos:

```
>> out = daqhwinfo;
>> out.InstalledAdaptors
```

Lo que dará como resultado en el caso de mi PC:

```
ans =
    'parallel'
    'winsound'
```

Una vez que Matlab ha reconocido el PP, ejecutamos:

```
>> daqhwinfo('parallel')
ans =
           AdaptorDllName: [1x50 char]
        AdaptorDllVersion: '2.7 (R14SP3)'
              AdaptorName: 'parallel'
               BoardNames: {'PC Parallel Port Hardware'}
        InstalledBoardIds: {'LPT1'}
    ObjectConstructorName: {''  ''  'digitalio('parallel','LPT1')'}
```

La última línea indica el nombre de la entrada digital: `digitalio('parallel','LPT1')`.

Como sabemos, el PP puede ser de entrada y salida. La siguiente figura muestra la distribución de pines del PP:

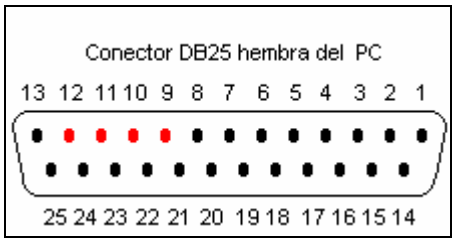


Fig. 48. Numeración de pines del PP.

En este conector:

- 8 pines son para salida de datos (bits de **DATOS**), y van desde el pin 2 (bit menos significativo) hasta el pin 9 (bit más significativo).
- 5 pines son de entrada de datos (bits de **ESTADO**). Estos pines son: 15, 13, 12, 10 y 11, del menos al más significativo.
- 4 pines son de control (bits de **CONTROL**). Tienen la característica de ser bidireccionales es decir que los puedes utilizar tanto de entrada como de salida. Estos pines son: 1, 14, 16 y 17, del menos al más significativo.

Sin embargo, configurando en la BIOS del PC (accesible en unos PCs con la tecla *F12*, en otros con la tecla *Del* o *Supr*, y a veces con *F10*) el puerto paralelo (LPT1) como de entrada y salida, es posible usar los pines del 2 al 9 como de entrada.

Para determinar cuantos pines podemos usar en el PP, ejecutamos:

```
>> parport = digitalio('parallel','LPT1');
>> hwinfo = daqhwinfo(parport)

hwinfo =
                AdaptorName: 'parallel'
                 DeviceName: 'PC Parallel Port Hardware'
                         ID: 'LPT1'
                       Port: [1x3 struct]
              SubsystemType: 'DigitalIO'
                 TotalLines: 17
    VendorDriverDescription: 'Win I/O'
        VendorDriverVersion: '1.3'
```

Como se puede ver en la información, 17 de los 25 pines del PP se los puede utilizar como I/O. Los restantes pines son *tierra*.

Una vez creada la entrada digital del PP, lo que sigue es asignar que pines serán para entrada y cuales para salida. Usamos la función *addline,* cuya sintaxis es:

```
>> dato2= addline(parport,0:7,'in'); %Para valores de entrada
>> dato = addline(parport,0:7,'out'); %Para valores de salida
```

Y se obtiene el dato de entrada con `dato3=getvalue(dato2)`. Se asigna el dato al puerto con: `putvalue(dato,255)`.

Con esta información se puede realizar la interfaz gráfica.

La parte de inicialización de datos se usa para colocar la imagen del PP:

```
function parallel_port_OpeningFcn(hObject,eventdata,handles,varargin)
handles.ada=0;
[x,map]=imread('db25.gif','gif');
%Representamos imagen en figura, con su mapa de colores
image(x),colormap(map),axis off,hold on
```

Como se puedo notar, se inicializa el manejador *handles.ada* en 0. Este manejador almacena el valor ingresado por el usuario para ser escrito en el PP. Se inicializa a 0 por que es el valor que presenta la interfaz en el *edit text* de salida al empezar a ejecutarse el programa.

El código del *callback* del *edit text* para ingresar el valor de salida al PP es:

```
function dato_out_Callback(hObject, eventdata, handles)
handles.ada=get(hObject,'String');
handles.ada=str2double(handles.ada);
guidata(hObject, handles);
```

El código del *callback* del botón ENVÍO es:

```
function envio_Callback(hObject, eventdata, handles)

if handles.ada>255 || handles.ada<0 || isnan(handles.ada)
    errordlg('Número fuera de rango','ERROR');
    set(handles.dato_out,'String','0');
    handles.ada=0;
end
diego=digitalio('parallel','LPT1');
dato=addline(diego,0:7,'out');
putvalue(dato,handles.ada);
guidata(hObject, handles);
```

La sentencia `if` se encarga de corregir errores de ingreso de datos. Los valores deben estar entre 0 y 255 y deben ser valores numéricos. En caso de ingresar un dato fuera de rango (como -9 o a) se presenta un mensaje de error y se escribe en el puerto el valor de 0.

Las líneas de código siguiente inicializan la entrada digital, asignan los pines de salida y escriben el valor en el PP.

El código del *callback* del botón RECEPCIÓN es:

```
function recibir_Callback(hObject, eventdata, handles)
judas=digitalio('parallel','LPT1');
dato2=addline(judas,8:12,'in');
dato3=getvalue(dato2);
dato4=binvec2dec(dato3);
set(handles.dato_in,'String',dato4);
guidata(hObject, handles);
```

Ya que *dato3* es un vector binario, *dato4* es el valor decimal de ese vector.

PROGRAMA *motorGUI*

El presente diseño contiene una breve introducción sobre los motores paso a paso así como el código de un programa desarrollado en Matlab 7.1 para controlar su velocidad y dirección de giro.

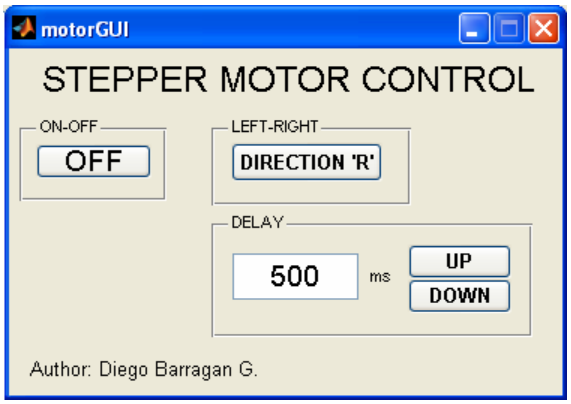


Fig. 49. Entorno del programa motorGUI.

Los motores paso a paso son ideales para la construcción de mecanismos en donde se requieren movimientos muy precisos. La característica de estos motores es poder moverlos un paso a la vez por cada pulso que se le aplique.

Matlab posee comandos sencillos para el control del puerto paralelo. Basta con crear la entrada digital del puerto y asignar que pines son de entrada y cuales de salida.

```
Command Window
>> ent=digitalio('parallel','LPT1');
>> dato=addline(ent,0:4,'out');
>> putvalue(dato,2);
```

El código anterior crea la entrada digital del puerto paralelo, asigna los pines 2 a 5 como salidas y escribe el valor decimal 2 (Matlab realiza automáticamente la conversión a binario) en el puerto.

Las condiciones iniciales del programa colocan a cero los pines del puerto paralelo:

```
diego= digitalio('parallel','LPT1');
dato= addline(diego,0:3,'out');
putvalue(dato,0);
```

Lo único que se debe programar en la interfaz gráfica es la captura del retardo, la dirección de giro y el encendido-apagado del motor.

La mayor parte del código se programa en el *toggle button* ON-OFF, cuyo campo *Tag* es *state*. Sin embargo, es necesario un código adicional para el texto del botón-interruptor de la dirección:

```
f=get(handles.direction,'Value');
if f==1
```

```
    set(handles.direction,'String',...
        'DIRECTION ''L''');
else
    set(handles.direction,'String',...
        'DIRECTION ''R''');
end
```

La siguiente figura muestra el campo *tag* de cada uno de los componentes:

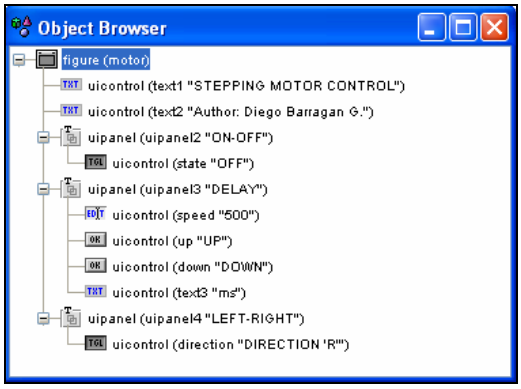


Fig. 50. Etiquetación (tag) de los componentes del programa.

La secuencia que se envía al motor es:

(A) Pin 5	(B) Pin 4	(C) Pin 3	(D) Pin 2	Decimal
1	1	0	0	3
0	1	1	0	6
0	0	1	1	12
1	0	0	1	9

La programación del botón de encendido es:

```
d=get(hObject,'Value');
if d==1
    set(handles.state,'String','ON');
    diego=digitalio('parallel','LPT1');
    dato=addline(diego,0:3,'out');
    g=1;
    while g
        e=get(handles.direction,'Value');
        if e==0
            mov=[3 6 12 9];
        else
            mov=[9 12 6 3];
        end
        delay=str2double(get(handles.speed,'String'))*1e-3;
        if delay<0 ||isnan(delay)
            errordlg('Time out of range','ERROR');
            delay=500;
            set(handles.speed,'String',500);
            set(handles.state,'String','OFF');
            set(handles.state,'Value',0);
            break;
        end
        if get(hObject,'Value')==0
            break
        end
```

```
        putvalue(dato,mov(1));
        pause(delay);
        if get(hObject,'Value')==0
            break
        end
        putvalue(dato,mov(2));
   delay=str2double(get(handles.speed,'String'))*1e-3;
        pause(delay);
        if get(hObject,'Value')==0
            break
        end
        putvalue(dato,mov(3));
   delay=str2double(get(handles.speed,'String'))*1e-3;
        pause(delay);
        if get(hObject,'Value')==0
            break
        end
        putvalue(dato,mov(4));
   delay=str2double(get(handles.speed,'String'))*1e-3;
        pause(delay);
    end
else
    set(handles.state,'String','OFF');
end
```

Para la conexión del motor se puede usar cualquiera de los siguientes esquemas:

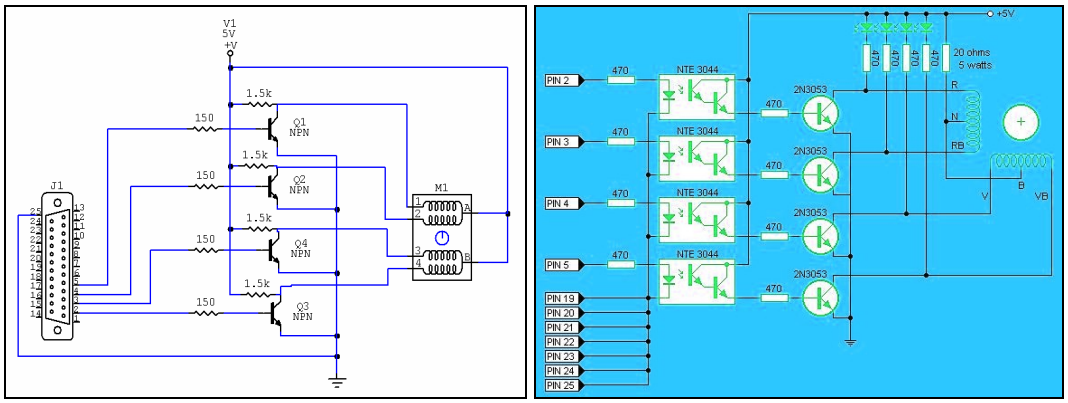


Fig. 51. Esquemas para la conexión del motor.

La siguiente figura muestra el modelo ensayado y funcionando:

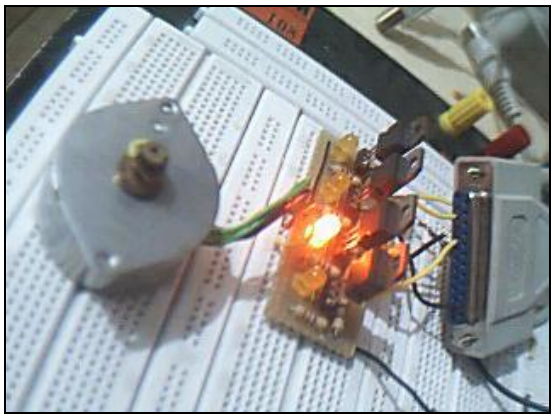


Fig.52. Circuito final.

Asimismo, la siguiente función controla el número de pasos del motor, velocidad y dirección de giro:

```
function motor(step,delay,direction)
%Controlling a stepper motor
%step must be integer.
%delay is in seconds.
%direction: 1 or 2.
%Author: Diego Orlando Barragán Guerrero
%diegokillemall@yahoo.com
warning off
if nargin==1
    delay=1;direction=1;
elseif nargin==2
    direction=1;
end
if isnan(step)||step<0
    error('Step value must be positive integer');
elseif direction~=1 && direction~=2
    error('Direction options: 1 or 2')
elseif isnan(delay)||delay<0
    error('Delay value must be positive integer')
end
step=ceil(step);
inout=digitalio('parallel','LPT1');
dato=addline(inout,0:3,'out');
if direction ==1
    sent=[3 6 12 9];
else
    sent=[9 12 6 3];
end
m=1;
for n=1:step
    putvalue(dato,sent(m));
    pause(delay);
    m=m+1;
    if m>4
        m=1;
    end
end
```

.EXE

MATLAB 7.1 posee un compilador que convierte nuestros programas en archivos .exe que pueden ejecutarse sin necesidad de abrir MATLAB o en otros ordenadores que no tengan instalado este software. Para ejecutar este compilador, es necesario tener instalado Simulink.

Ejecutamos en la ventana de comandos el siguiente código:

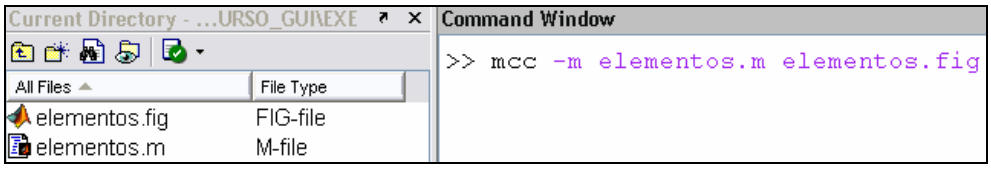


Fig. 53. Comando mcc para el compilador.

El compilador creará los siguientes archivos. El último archivo de esta lista es el ejecutable.

Fig. 54. Archivos resultantes de la compilación.

Para instalar una aplicación en un ordenador que no tenga Matlab:

1. Copiar el instalador del RunTime de un ordenador con Matlab:

<matlabroot>\toolbox\compiler\deploy\win32\MCRInstaller.exe

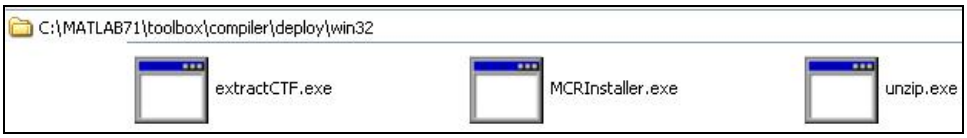


Fig. 55. Instalador del RunTime de Matlab.

2. Desempaquetar los instaladores, haciendo doble clic en MCRInstaller (ya en el ordenador sin Matlab).

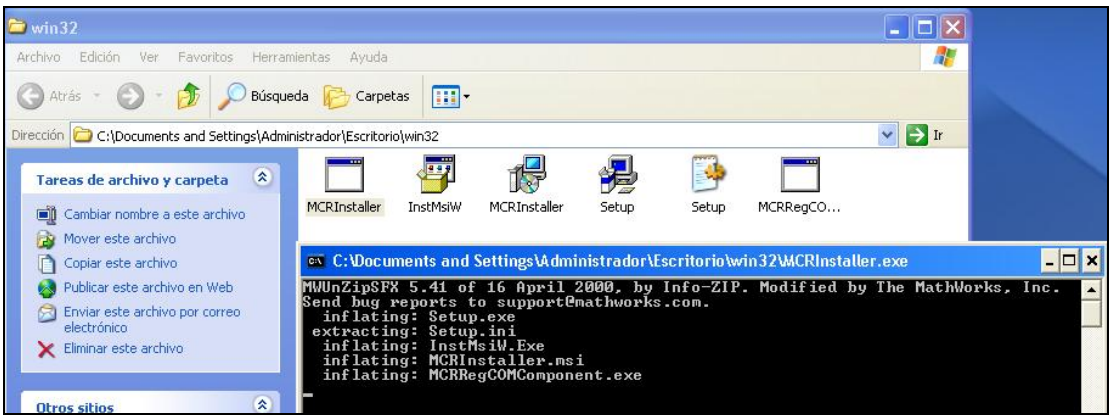


Fig. 56. Extracción del paquete de instalación.

3. En el ordenador de destino:

Instalar MCRInstaller en C:\MCR (por ejemplo).

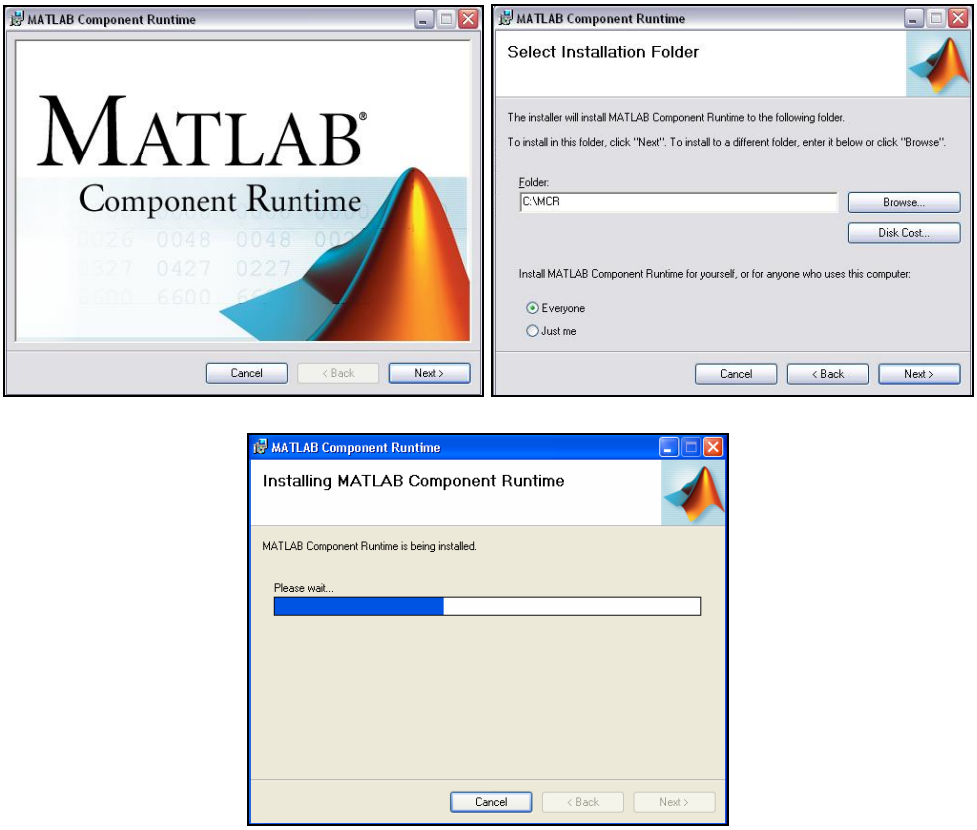


Fig. 57. Instalación del componente RunTime en el ordenador sin Matlab.

Luego de la instalación del RunTime, asegurarse de que *C:\MCR\v73\runtime\win32* esté en el PATH. Para esto, hacer click derecho en el ícono *Mi PC*, seleccionar *propiedades*, *opciones avanzadas*, *variables de entorno*, *variables de sistema.*

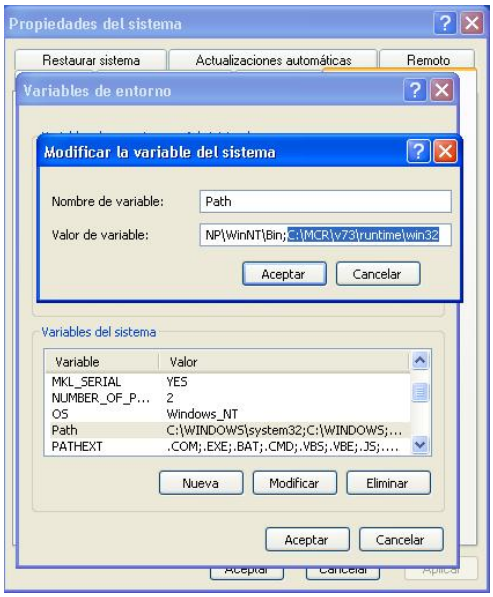


Fig.58. Path del SO.

En caso de no encontrarse agregar al path: *C:\MCR\v73\runtime\win32*

Copiar elementos.exe y elementos.ctf al directorio de la aplicación del CPU que no tiene Matlab. Al abrir el archivo .exe, aparecerá la pantalla del DOS y luego el programa:

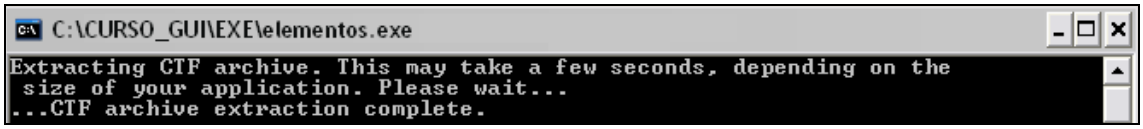


Fig. 59. Extracción de los archivos CTF.

TIPS GUIDE

Esta sección estará dedicada a exponer algunas características de las interfaces gráficas.

Cambiar color de fondo de una interfaz

En la parte de las condiciones iniciales se coloca:

```
set(gcf,'Color',[0.5 0.9 0.5])
```

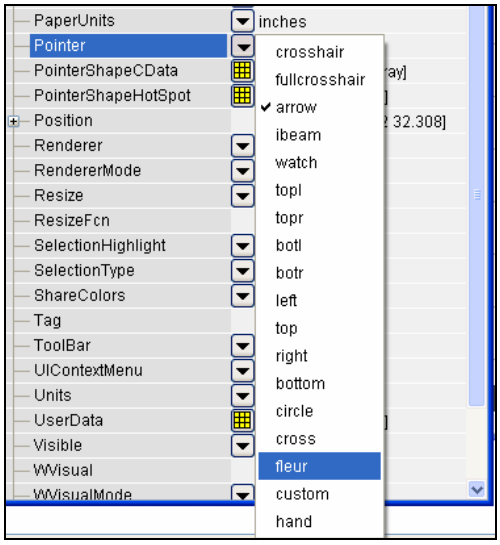
La matriz [0.5 0.9 0.5] es la matriz *RGB* (red, green, blue). Cada uno de los elementos de esta matriz va de 0 hasta 1.

Cambiar el tamaño de la interfaz

Haciendo doble click en el área de diseño de la interfaz, activar el parámetro *resize*.

Cambiar el cursor de la interfaz

Haciendo doble click en el área de diseño de la interfaz, cambiar el parámetro *pointer*.



Cambiar el color de los botones en Windows XP

Algunas veces el cambio de color de los botones no da resultado en Windows XP. Para conseguir esto, simplemente se cambia la apariencia a Windows clásico.

Cambiar de nombre a una interfaz

Aunque esto parezca sencillo, cuando se cambia el nombre a una GUI con *rename* se produce un error. Esto es consecuencia de que el nombre de la interfaz debe ser el mismo de la función principal.

Para cambiar el nombre a la interfaz simplemente se usa *Save As* en el archivo *.fig*.

Obtener el código ASCII usando KeyPressFcn

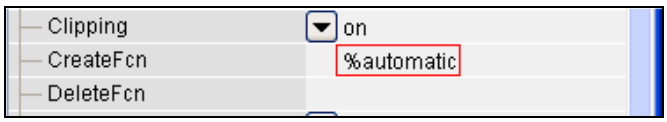
Luego de crear la función *KeyPress*, añadir la siguiente función:

```
a=double(get(gcbf,'Currentcharacter'));
```

Con esta función se captura el código ASCII en la variable *a*.

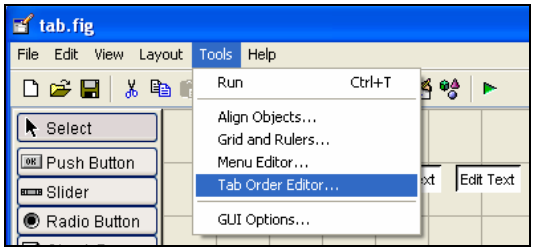
Eliminar código excesivo

Por lo general las funciones *CreateFcn* no se usan en una interfaz de usuario (*CreateFcn* define las condiciones iniciales de un objeto, pero las podemos establecer en la función de apertura del programa). Para que no se genere el código correspondiente a estas funciones, simplemente antes de guardar por primera vez la interfaz, en la parte de *CreateFcn* eliminar la palabra *%automatic* y el código no se generará.

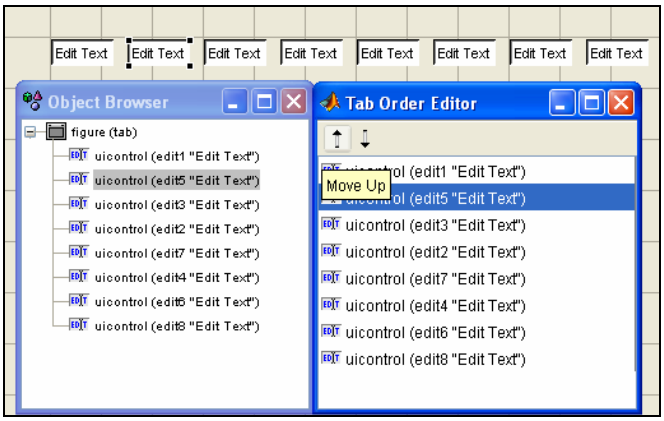


Arreglar el orden del TAB

Para ordenar los saltos del TAB en la GUI, se usa la herramienta *Tab Order Editor*.



Y para mayor facilidad usar también el *navegador de objetos*.



Colocar un ícono de ayuda en un push button (?)

El botón está etiquetado como help_button

```
s= load('helpicon')
set(handles.help_button, 'Cdata',siconRGB)
```

Editar dentro de un mismo string

```
set(handles.angulo,'string',['Angulo= ',num2str(ang),' Grados']);
```

Escribir datos a una hoja de excel

```
xlswrite('EXCEL',[1; 2; 3],'Hoja1','A1');
```

MATLAB *MANÍA*

Esta sección contiene links de los programas publicados en www.mathworks.com.

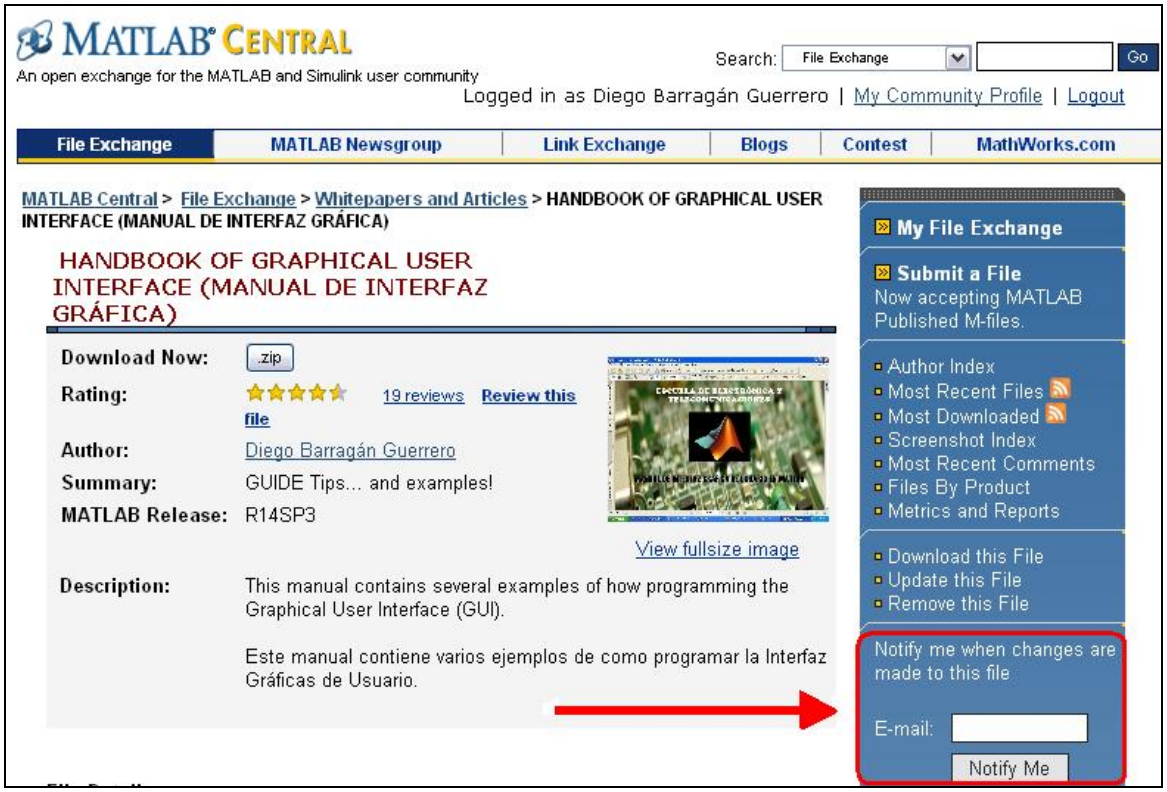
Funciones que simulan algunas modulaciones digitales:

http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=14410&objectType=FILE

Manual de interfaz gráfica. Como este tutorial se actualiza cada mes, se recomienda descargarlo de este link:

http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=12122&objectType=FILE

O puedes recibir en tu correo el aviso cuando el manual se ha actualizado, simplemente inscribiéndote donde indica la flecha roja de la figura:



Interfaz gráfica de usuario que simula modulaciones digitales.

http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=14328&objectType=FILE

Simulación en simulink de la modulación BPSK.

http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=12691&objectType=FILE

Lectura y escritura del puerto paralelo.

http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=12676&objectType=FILE

Interfaz gráfica que permite diseñar filtros digitales:

http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=13376&objectType=FILE

Simulación de modulación AM.

http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=12240&objectType=FILE

Simulación de modulaciones digitales usando System Generator de Xilinx.

http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=14650&objectType=FILE

Interfaz gráfica que simula algunos códigos de línea.

http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=13498&objectType=FILE

Funciones que simulan algunos códigos de línea.

http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=13553&objectType=FILE

Control de un motor paso a paso.

http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=16439&objectType=FILE

Interacción entre GUIDE y Simulink.

http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=14359&objectType=FILE

ACERCA DEL AUTOR

Diego Orlando Barragán Guerrero (2/Sep/84) es estudiante Electrónica y Telecomunicaciones. Actualmente trabaja en la investigación de Procesamiento Digital de Imágenes en Matlab y *Software-Defined Radio*. Ha publicado algunos programas en www.mathworks.com sobre programación de interfaces gráficas, diseño de filtros, códigos de línea, modulación AM, modulación digital…

Ha dictado cursos de Matlab y Simulink en la Universidad Técnica Particular de Loja (UTPL) y en la Universidad Técnica de Ambato (UTA).

Es hincha del EMELEC, el "Che" Guevara, la buena Literatura y el Heavy-Metal.

Release: 2007-12-13

DE ECUADOR PARA EL MUNDO