Universidad de Buenos Aires

Facultad de Ingeniería

Taller de Programación III

"Estación de Enseñanza de Control Automático de Niveles de Líquido en Tanques"

Manual de Programación

Autores:

Ezequiel Di Donato (75.781)

Pablo D. Roca (82.904)

Índice de contenido

Introducción

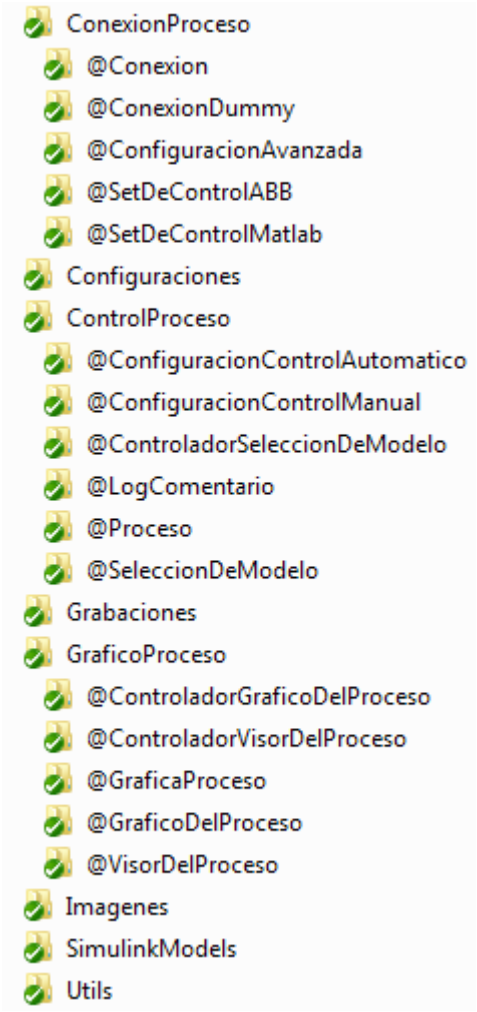
El desarrollo del sistema testbed-control-procesos fue hecho enteramente en Matlab 6.5 R13 utilizando las características de orientación a objetos que el entorno ofrece, si bien se las conoce limitadas.
La interfaz a usuario también está desarrollada en MATLAB puro, lo que garantiza máximo rendimiento especialmente al ejecutar funciones en paralelo, como el muestreo del proceso y la actualización de las pantallas reflejando los cambios y atendiendo los eventos del usuario.

Convención del Código

Nomenclatura en construcción clases y métodos

El código se organiza en métodos distribuidos en clases. A su vez las clases se agrupan por cohesión, es decir aquellas que tienen funcionalidad similar y están más acopladas o dependientes entre sí, pertenecen a un mismo paquete.
Cada clase pretende ser una unidad de funciones y datos estrechamente relacionados con una interfaz clara de uso por el resto de las clases.
Los paquetes están nombrados en *PascalCase*, es decir cada primera letra de cada palabra en su nombre comienza con mayúscula.
Las clases también son directorios, conforme al entorno de programación utilizado, la nomenclatura utilizada es *PascalCase* al igual que en los paquetes. En este caso, y para que el entorno la reconozca como clase, su nombre debe comenzar con un símbolo particular: @ (arroba).
Así, todos los métodos de una clase son archivos ejecutables matlab de extensión implementando una función. Como particularidad de este esquema, el nombre de cada función debe coincidir con el del archivo.

Estructura del Árbol



Una diferencia notable con los lenguajes fuertemente orientados a objetos, es que la referencia a la instancia del objeto debe pasarse y/o devolverse como parámetro de la función que implementa sus métodos, sobre todo si desean obtenerse o modificarse el estado del objeto. No es posible acceder a la instancia del objeto desde sus métodos directamente como en otros lenguajes.
Finalmente los nombres de los atributos y métodos también son *camelCase*, similar al *PascalCase* pero su letra inicial es minúscula. Así se diferencia de los nombres de paquetes y clases.
En el siguiente ejemplo, se muestra un ejemplo del pasaje de la instancia del objeto como parámetro y su modificación lo que necesariamente implica devolverla como valor de retorno del método.

```
function self = agregarComentario( self , textoComentario)
    [self.proceso, comentario] = agregarComentario(self.proceso,
textoComentario);
    w = getWindow('VisorDelProceso');
    agregarComentario(w.vista, comentario);
end
```

El código correspondiente a las interfaces gráficas en cambio no está implementado como clases, sino que cada pantalla se compone de dos archivos: uno el de la definición de la pantalla de extensión .fig y el otro correspondiente a las funciones de atención a eventos de extensión .m.
Los pares de archivos correspondientes a interfaces al usuario se encuentran juntos dentro del paquete y al mismo nivel que los directorios de clases con las cuales tengan más dependencias. Sus nombres son *PascalCase* como las clases y paquetes y terminan con el sufijo “GUI”.

Ámbito de las variables

El ámbito de las variables se ha limitado a su uso y necesidad, así todas las variables cuyo uso está limitado a una única función o método, se encuentra definida dentro de la misma.
Los objetos principales son normalmente utilizados por varias pantallas, por lo que este tipo de variables así como otras de uso compartido a lo largo del código se encuentran organizadas en una única estructura global, disponible y accesible desde cualquier ámbito de la aplicación.
De esta manera se evitan largas definiciones de funciones con muchos parámetros y pasaje extenso de argumentos en cada una de sus llamadas. Cabe destacar en relación a esto que el **pasaje de parámetros en Matlab es siempre por valor**, por lo que toda estructura o variable compleja o incluso objeto que deba ser modificado deberá pasarse por valor en la función (o incluso método de esa clase) correspondiente y ser devuelto por dicha función o método, de otra manera los cambios no serán visibles al resto de la aplicación.
Las variables de uso global están definidas en el módulo matlab que inicia la aplicación testbed_controldeprocesos.m en el paquete raíz. Las mismas se enumeran a continuación.

Variables Globales

- **directorioInicio**: indica el directorio desde donde se inicio la aplicacion para poder hacer referencias relativas a otros archivos y módulos desde el código.
- **configuracionAvanzada**: estructura que contiene los valores de uso global en toda la aplicación como ser puerto de conexión, velocidad, configuración por defecto y límites de ejes para los gráficos.
- **conexion**: estructura que contiene los datos de una conexión activa.

- **SetDeControl**: referencia al set de control o modo en que se controlará la planta, según la forma elegida: automático (PID implementado en Simulink) o manual (por el usuario).
- **windowHandles**: arreglo de referencias a las interfaces gráficas (pantallas) así como a los objetos controladores de las mismas que las lanzan y actualizan según corresponda.
- **graficoProcesoZoom**: estructura al gráfico ampliado de un proceso.

Estructura General

Estructura de directorios

Directorios de Recursos

- **Imagenes**: en este directorio se almacenan las imágenes que se utilizan en la interfaz gráfica como íconos de los botones y demás controles.
- **Configuraciones**: en este directorio se almacenan las distintas configuraciones de valores para el control de un proceso en forma predeterminada, sin embargo el usuario puede decidir donde las guardará.
- **Grabaciones**: en este directorio se almacenan por defecto los grabaciones de los procesos, pero esto puede ser cambiado por el usuario.
- **SimulinkModels**: en este directorio se ubica el modelo PID implementado en Simulink para el control automático. Simulilnk es utilizado por el software pero no mostrado al usuario.

Directorios de Paquetes

- **ConexionProceso**: paquete que contiene las clases correspondientes a la conexión de la placa adquisidora de datos.
- **Control** Proceso: paquete que contiene las clases que efectúan el control de un proceso, utilizando la conexión, así como el código de pantallas que permite elegir el modo de conectar.
- **GráficoProceso**: paquete que contiene las clases correspondientes a la presentación de la información de los procesos, ya sean grabaciones recuperadas o bien control y supervisión con una conexión activa.
- **Utils**: paquete de funciones de utilidad general, usadas a lo largo del proyecto.

Comunicación con RIAC

La comunicación desde matlab con la placa adquisidora RIAC es llevada a cabo cargando un control ActiveX que viene junto con el controlador de la placa y desde luego debe estar instalado en el sistema operativo.

En el paquete Conexion, clase Conexion, método crearYConectar se encuentra la interacción desde matlab con dicho control ActiveX. El mismo es cargado mediante la siguiente instrucción:

```
self.placa = actxserver('RiacQXControl.RiacQX');
```

donde self es el objeto Conexion y placa el atributo que mantiene el estado de la misma. Desde luego, la conexión con este dispositivo, requiere ciertos parámetros como ser la velocidad de comunicación en bauds/segundo, el número de puerto, que son obtenidos del objeto global ConfiguracionAvanzada.
A continuación se muestran los ejemplos para obtener y enviar información.

Obtener el nivel del líquido: paquete ControlProceso, clase Conexion, método obtenerVivel.

```
function [nivel ] = obtenerMuestra( self )
%obtenerMuestra Obtiene una muestra de la placa para el canal de
la
%conexion
    nivel = AnalogicInput(self.placa, self.canalSensor)/1298*3;
    if ~strcmp(self.placa.ErrNumber, 'NoError')
          error('Conexion:crearYConectar', sprintf('Ha ocurrido
un error desconocido durante la adquisicion de datos de la placa
RIAC. Codigo de error: (%s)', self.placa.ErrNumber));
    end
end
```

Establecer el valor del actuador: paquete ConexionProceso, clase Conexion, método establecerValorActuador.

```
function [nivel ] = obtenerMuestra( self )
%obtenerMuestra Obtiene una muestra de la placa para el canal de
la conexion
    nivel = AnalogicInput(self.placa, self.canalSensor)/1298*3;
    if ~strcmp(self.placa.ErrNumber, 'NoError')
          error('Conexion:crearYConectar', sprintf('Ha ocurrido
un error desconocido durante la adquisicion de datos de la placa
RIAC. Codigo de error: (%s)', self.placa.ErrNumber));
    end
end
```

Comunicación con módulos simulink

Como se mencionó, el control automático desde el software está implementado como un lazo tipo PID, en Simulink. De manera que al intentar controlar desde Matlab el proceso conectado, es necesario cargar la definición de dicho lazo utilizando Simulink.
Para ello, Matlab ofrece la función **load_sym** que permite cargar un modelo desarrollado en simulink en forma programática, sin necesidad de solicitar ninguna intervención al usuario.
La función **load_sym** abriría una ventana en Simulink, visible al usuario con el modelo allí implementado, lo que no es deseable para esta aplicación, dado que se desea abstraer al usuario del manejo específico de Matlab. No obstante esta misma función permite efectuar la carga de Simulink en segundo plano de manera que no es visible al usuario, pero al mismo tiempo permanece corriendo junto con el módulo Simulink de Matlab, de manera que programáticamente se puede interactuar como si estuviera en primer plano.
La carga de Simulink en *background* se encuentra en el paquete ControlProceso, clase SetDeControlMatlab, en el metodo constructor SetDeControlMatlab.

Manejo de paralelismo: empleo de timer

Debido a la interacción que ofrece la aplicación y la característica de poder visualizar en tiempo real el proceso al cual se está conectado, es necesario un mínimo manejo de paralelismo, de forma tal de mantener el hilo de ejecución principal de la aplicación respondiendo a los eventos de usuario, sin dejar de obtener las muestras del proceso actual.
Para ello, el muestreo del proceso que permite la placa adquisidora de datos es llevado a cabo por timers de Matlab. Los mismos constituyen algunas ventajas con respecto a otras posibles soluciones, como empleo de hilos o procesos lanzados adicionalmente que escriban en archivos intermedios. Los timers están provistos por la misma plataforma de trabajo, por consiguiente, abstraen al programador de los problemas de concurrencia y sincronismo de código, permitiendo enfocar el código en la resolución del problema esencial.
Los timers se encuentran declarados en el paquete ConexionProceso, clase SetDeControlMatlab, método constructor: SetDeControlMatlab.

Timers Utilizados:

- Actualizar el gráfico en la pantalla Visor del Proceso y los valores en el diagrama en bloques: ejecuta la funcion agregarMuestra, dentro del paquete GraficaProceso.
- Actualizar los valores en la pantalla Grafica del Proceso: ejecuta la agregarMuestraGrafico, dentro del paquete GraficaProceso.

Detalle por Paquete

Paquete ConexionProceso

Clase Conexion

Esta clase encapsula la comunicación con la placa adquisidora de datos, mediante el control ActiveX “RiacQXControl.RiacQX”.

Método crearYConectar

Este es uno de los métodos más importantes y centrales de la clase y del trabajo en general, dado que establece la comunicación del software con la placa adquisidora de datos, mediante el la invocación al control ActiveX “RiacQXControl.RiacQX”, como se muestra a continuación.

```
self.placa = actxserver('RiacQXControl.RiacQX')
```

Luego utiliza la estructura devuelta por la instrucción anterior para establecer y probar la conexión, una líneas de código posteriores:

```
nivel = AnalogicInput(self.placa, self.canalSensor);
```

Método obtenerNivel

Obtiene el nivel actual del líquido en el tanque mediante la estructura retornada por el control ActiveX, utilizando sus campos y normaliza el nivel a la escala.

```
nivel = AnalogicInput(self.placa, self.canalSensor)/1298*3;
```

Métodos obtenerValorActuador y establecerValorActuador

Trabajan con el valor del actuador así como en el caso del nivel, la diferencia se nota al escribir valores mediante el control ActiveX en la placa adquisidora, en este caso se usa un AnalogOutput como sigue:

```
AnalogicOutput(self.placa, self.canalActuador, valor);
```

Clase ConexionDummy

Esta clase es usada como alternativa, cuando se desea reproducir en el tiempo las muestras de una simulación, solo que en lugar de existir una conexión real a la planta toma las muestras de un archivo.

Métodos

Presenta la misma interfaz que la anterior, pero lee de un archivo en lugar de la placa adquisidora.

Clase ConfiguracionAvanzada

Esta clase se encarga de leer y escribir los parámetros de la aplicación en un archivo y mantener sus valores durante toda la ejecución del programa para brindarlos cuando las distintas clases y métodos de pantallas lo soliciten.

Métodos leer y guardar

Este método simplemente lee del archivo de configuración los valores de los parámetros que se usarán globalmente en el resto del programa. Debido a que Matlab permite leer y guardar estructuras así como cualquier otra variable usando un formato propio, cuya extensión es usualmente **.mat**, la lectura y escritura de todos estos parámetros resultan sencillas utilizando dos simples instrucciones: **load** y **save** existentes en Matlab.

Para guardar la configuración:

```
save(strcat(directorioInicio,'/configuracionAvanzada.mat'),
'self');
```

Para recuperar la configuración:

```
file = strcat(directorioInicio,'/configuracionAvanzada.mat');
  if exist(file, 'file')
    fileSelf = load(file);
```

En ambos casos nótese el uso de la variable global directorioInicio, asignada al comenzar la aplicación, como se dijo anteriormente es para permitir al resto del código independizarse de la ubicación donde el programa fue instalado, es decir esta variable hace de prefijo para las rutas relativas de la aplicación en todo el código.

Otros Métodos: "getters" y "setters".

El resto de los métodos son sencillamente para acceder o asignar cada uno de los atributos del objeto ConfiguracionAvanzada, global durante toda la vida de la aplicación.

Clase SetDeControlMatlab

Esta clase se encarga de cargar el módulo simulink que implementa el PID, desde la presente aplicación e iniciar los timers.

Constructor SetDeControlMatlab

En el constructor se construyen los timers con el siguiente código:

```
timerFunction = sprintf('fprintf(''Simulation
Started\\n'');load_system(''%s'');', modeloSimulink);
timerFunction = strcat(timerFunction, 'global setDeControl;
setDeControl = aplicarParametrosConfiguracion(setDeControl);');
timerFunction = strcat(timerFunction, sprintf('set_param(''%s'',
''SimulationCommand'', ''start'');', modeloSimulink));
```

Método iniciar

Este método inicia los timers definidos en el constructor, al comienzo del control de la planta.

```
start(self.timer)
```

Método modificar

Modifica los parámetros de control en el modelo en Simulink para cambiar el comportamiento del sistema, según el usuario lo requiera. La sentencia que lleva a cabo dicha función es:

```
set_param(strcat(self.modeloSimulink, '/', parametro),propiedad,
num2str(valor));
```

Método cerrar

Termina la comunicación con la placa adquisidora y cierra el módulo Simulink cargado en background, detiene asimismo el timer, el código correspondiente es:

```
set_param(self.modeloSimulink, 'SimulationCommand', 'stop');
delete(self.timer);
```

Pantalla ConexionAvanzadaGUI

Es la implementación de la pantalla que permite el cambio de configuración general de la aplicación y valores por defecto. Esta es una pantalla, no una clase, por lo tanto no posee métodos sino funciones de atención a eventos, provenientes de Matlab.

Función ConfiguracionAvanzadaGUI_OpeningFcn

Esta función se llama al iniciar la pantalla correspondiente y debe agregarse el código de inicialización de la misma. En este caso, lo que se hace es cargar los valores actuales de configuración de la aplicación en cada uno de los campos de texto.
A modo de ejemplo se proveen una líneas:

```
% --- Executes just before ConfiguracionAvanzadaGUI is made
visible.
function ConfiguracionAvanzadaGUI_OpeningFcn(hObject, eventdata,
handles, varargin)
    set(handles.edit_Velocidad, 'String',
getVelocidad(configuracionAvanzada));
    set(handles.edit_Puerto, 'String',
getPuerto(configuracionAvanzada));
    set(handles.edit_Periodo, 'String',
getPeriodo(configuracionAvanzada));
```

Funciones de Eventos sobre los Objetos de la Pantalla

Cuando el usuario realiza o modifica el valor de algún control gráfico, se llama a la rutina que atiende dicho evento, la convención de nombres es impuesta por Matlab, por ejemplo, al editar el valor del máximo nível, la función correspondiente será:

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

Función para el Evento Click en Aceptar

Al aceptar las modificaciones hechas, la rutina que atiende al evento se encarga de validar uno por uno los campos y luego invocando al método guardar de la clase ConfiguracionAvanzada persiste los mismos. El código al respecto es:

```
function pushbutton_Aceptar_Callback(hObject, eventdata, handles)
global configuracionAvanzada
  validado = validarConfiguracion(handles);
  if (validado)
    configuracionAvanzada =
recolectarValores(configuracionAvanzada, handles);
    configuracionAvanzada = guardar(configuracionAvanzada);
    close(handles.output);
  end
```

Notar como al invocar el método guardar es necesario pasar la variable objeto global configuracionAvanzada y reasignarla como resultado. Esto ejemplifica como se dijo en las convenciones de código, al limitado manejo de objetos del lenguaje utilizado y la carencia de pasaje de parámetros por referencia. La asignación del resultado al mismo objeto al que pertenece el método es para que los cambios sean realmente reflejados en dicha variable.

Paquete ControlProceso

Clase ConfiguracionControlAutomatico

Esta clase se encarga de recuperar y almacenar los valores de configuración para el control del proceso en modo automático PID.

Contructor ConfiguracionControlAutomatico

En el constructor se inicializan los valores de ajuste del control como ser kp, ki, etc. y se validan que estén dentro de los rangos permitidos.

Métodos recuperar y guardar

Estos métodos leen el conjunto de parámetros desde un archivo o bien almacenan los valores de configuración actuales en un archivo. Utiliza las funciones load y save de matlab, como se muestra a continuación:

```
function self = restaurar( self, archivo )
    data = load(archivo);
function self = guardar( self , archivo)
    save(archivo, 'self');
```

Métodos: "getters" y "setters".

Existen muchos otros métodos cuya función es asignar o bien devolver el valor de los parámetros de configuración, los mismos están nombrados aludiendo al dato que manejan. Estos métodos se utilizan cuando el usuario desde la interfaz gráfica, decide modificarlos.

Clase ConfiguracionControlManual

Esta clase se encarga de recuperar y almacenar los valores de configuración para el control del proceso en modo manual.

Métodos

Los métodos de esta clase son muy similares a ConfiguracionControlAutomatico, exceptuando los métodos getters y setters, dado que el único parámetro de control a variar, es la salida del actuador, no existiendo otros tales como constantes derivativas, de integración, etc.

Clase ControladorSeleccionDelModelo

Esta clase se encarga de iniciar la conexión, crear la instancia apropiada de la clase SetDeControlAutomatico o SetDeControlManual en el modo elegido por el usuario, cargando la configuración apropiada y lanzando la pantalla de visualización del proceso. También es invocado para recuperar procesos grabados desde archivos.
Depende e invoca a las clases anteriores e inicia la pantalla VisorDelProcesoGUI.

Método conectar

Este método crea la conexión según el tipo de control elegido automático o manual y abre la ventana Visor del Proceso. También abre la ventana de reproducción de un proceso grabado.
La secuencia que sigue es la siguiente:

1. Crea una conexión
2. Inicializa la conexión
3. Crea un objeto de la clase Proceso
4. Crea el objeto de la clase SetDeControlManual o SetDeControlAutomatico
5. Inicializa los parámetros de control
6. Abre la ventana Visor del Proceso

Método cargarProceso

Carga un proceso desde archivo y abre la pantalla Grafico del Proceso.
Su código es sencillo y autoexplicativo.

```
proceso = Proceso;
proceso = leer(proceso, archivo);
GraficoDelProcesoGUI(proceso);
```

Clase LogComentario

Esta clase mantiene los comentarios ingresados por el usuario durante una grabación y su relación a las muestras para ser grabados y/o recuperados en/de las grabaciones de procesos.

Constructor LogComentario

Crea un objeto comentario e inicializa sus atributos. Su firma es la siguiente:

```
LogComentario( numero, texto, instante, valor)
```

Los comentarios se numeran en orden creciente de manera de poder marcarlos sobre el mismo gráfico por número. El texto es una cadena de caracteres libre ingresada por el usuario, instante es el momento en el tiempo en que se ingresó (abscisa) mientras que el valor es la ordenada en el gráfico.

Métodos setters y getters

Son los métodos que permiten acceso a los atributos mencionados en el constructor.

Método toString

Este método devuelve una línea de texto con el formato adecuado para mostrarlo en la pantalla Visor del Proceso, una vez que el comentario se ha agregado.

Clase Proceso

Esta clase mantiene la información de las muestras del proceso en curso o bien el que se grafica desde una grabación.

Constructor Proceso

En el constructor simplemente se inicializan los atributos con valores iniciales y en muchos casos vacío como el vector de muestras.

Método agregarCambioConfiguracion

En el proceso se guardan todos los datos como para poder recuperarlo e incluso reproducirlo. Por lo tanto, al controlar un proceso es necesario también almacenar los cambios en los parámetros de control (Kp, Ki, salida del actuador (modo manual), etc.) y reflejarlos en la reproducción.

Este método guarda en el proceso los cambios en dichos parámetros, que el usuario pudiere hacer durante la grabación.

Método agregarComentario

Este método agrega un comentario (ver clase LogComentario) al proceso, de modo de poder guardarlos al finalizar al grabación y recuperarlos al abrir un proceso existente.

Método agregarMuestra

Este método agrega una nueva muestra del proceso en curso.

Método cancelarGrabacion

Este método cambia el estado del proceso para reflejar que no está siendo grabando.

Método cerrar

Detiene la grabación actual e invoca al método guardar para almacenar la información del proceso capturada hasta el momento.

Método grabar

Este método modifica el estado del proceso para reflejar que se está grabando a partir de ese momento, es decir se invoca al iniciar la grabación y almacena el instante de inicio para no guardar valores previos.

Método guardar

Este es el método que finalmente realiza las acciones necesarias para almacenar los datos del proceso hasta el momento, en un archivo. Estos datos incluyen cambios en la configuración de los parámetros de control, comentarios, y las muestras de la gráfica del proceso. Asimismo cambia el estado del objeto a modo no grabación.

Método leer

Es la contraparte del método guardar. Lee el contenido completo de un proceso almacenado en un archivo y lo carga en el objeto proceso para que pueda ser mostrado en un gráfico.

Método normalizarMuestra

Este método realiza el cálculo apropiado sobre una muestra para obtener el valor a graficar, según los límites establecidos en el objeto global ConfiguracionAvanzada.

Métodos setters y getters

Son los que permiten acceso a la información de un proceso como ser los comentarios, muestras, a fin de poder presentarlos en los gráficos y pantallas, tanto al estar conectado a la placa adquisidora como cuando se recupera la información de disco.

Muchos de estos métodos tienen dos versiones, una de ellas agrega el sufijo “normalizado”, indicando que se aplica el método de normalización según la escala configurada al dato a devolver.

Pantalla SeleccionDelModeloGUI

Es la pantalla inicial que despliega las distintas opciones de conexión o visualización de procesos que estén grabados.

Función SeleccionDeModeloGUI_OpeningFcn

Según la documentación de programación del Matlab, es la función que se ejecuta al abrir la pantalla. Su firma y encabezado es generado automáticamente por Matlab, como se muestra a continuación:

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

En esta función se inicializa la pantalla que permite seleccionar el modo de conexión, es decir la inicial, inicializa los valores por default, carga las imágenes para los iconos de los botones

Otras Funciones

A continuación se presentan los encabezados de las funciones más importantes de esta pantalla, cuyo objetivo es atender los eventos que el usuario genera en esta pantalla, presionando un botón, seleccionando una opción, o ejecutando un item de menú.

```
function ConfiguracionAvanzada_Callback(hObject, eventdata,
handles)
% hObject handle to ConfiguracionAvanzada (see GCBO)
% handles structure with handles and user data (see GUIDATA)
```

```
function Salir_Callback(hObject, eventdata, handles)
% hObject handle to Salir (see GCBO)
% handles structure with handles and user data (see GUIDATA)
```

```
function Grabacion_Callback(hObject, eventdata, handles)
% hObject handle to Grabacion (see GCBO)
% handles structure with handles and user data (see GUIDATA)
```

```
function AbrirGrabacion_Callback(hObject, eventdata, handles)
% hObject handle to AbrirGrabacion (see GCBO)
% handles structure with handles and user data (see GUIDATA)
```

```
% --- Executes on button press in rdManual.
function rdManual_Callback(hObject, eventdata, handles)
% hObject handle to rdManual (see GCBO)
% handles structure with handles and user data (see GUIDATA)
```

```
% --- Executes on button press in rdAutomaticoMatlab.
function rdAutomaticoMatlab_Callback(hObject, eventdata, handles)
% hObject handle to rdAutomaticoMatlab (see GCBO)
% handles structure with handles and user data (see GUIDATA)
```

```
% --- Executes on button press in btnConectar.
function btnConectar_Callback(hObject, eventdata, handles)
% hObject handle to btnConectar (see GCBO)
% handles structure with handles and user data (see GUIDATA)
```

Paquete GraficoProceso

Clase ControladorVisorDelProceso

Esta clase es el controlador encargado de mostrar y visualizar la pantalla VisorDelProcesoGUI.

Método iniciar

Este método inicia los timers correspondientes con el fin de obtener las muestras y almacenarlas en la clase Proceso, para luego sean graficadas, almacenadas en archivo, etc.

Método comenzarGrabación

Modifica el estado del objeto proceso para dejarlo en modo grabación y muestra un mensaje al usuario que la grabación ha comenzado.

Método finalizarGrabacion

Finaliza la grabación actual invocando los métodos apropiados en la clase Proceso y efectúa el almacenamiento en archivo desde el momento en que se comenzó la grabación. Esta información está en la clase Proceso, como se explicó anteriormente.

Métodos modificarParámetros

Existen una serie de métodos denominados de esta manera por cada uno de los parámetros de control que pueden ser modificados por el usuario durante el control del proceso al que está conectado.
Estos métodos validan que el nuevo valor del parámetro sea correcto y efectúan el cambio invocando la clase SetDeControl.

Método procesarMuestraGrafico

Toma el valor de la muestra, obtiene los valores normalizados según la escala utilizando la clase Proceso y actualiza el gráfico que muestra el proceso.

Pantalla VisorDelProcesoGUI

Pantalla que se abre cuando se inicia una conexión o reproducción desde una grabación. Muestra elementos visuales para variar parámetros de control, la gráfica del proceso, un esquema del modelo de control del tanque y, en modo grabación, el listado e ingreso de comentarios.

Función VisorDelProcesoGUI_OpeningFcn

Esta función se ejecuta al abrir la pantalla. Posee el código de inicialización de todos los controles de la pantalla así como la recepción de parámetros de estructura que permiten interactuar con el objeto Figure de Matlab.
A continuación se presenta la firma de dicha función generada automaticamente por Matlab al crear una nueva pantalla.

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

Funciones Callback

Existen varias funciones denominadas por el nombre de un control gráfico en la pantalla, muchos de ellos cajas de texto que comienzan con el prefijo txt_. Estas funciones se ejecutan cuando alguno de estos controles gráficos con modificados (por ejemplo se asigna un nuevo valor a la constante derivativa) de manera de poder realizar los cambios en el modelo PID en simulink y que dichos cambios tengan efecto en el control del proceso.

Función IniciarGrabacion_Callback

Esta función se ejecuta cuando el usuario seleccionar Comenzar Grabación del menu e invoca a la clase ControladorVisorDeProceso.comenzarGrabacion.

Función FinalizarGrabacion_Callback

Esta función se ejecuta cuando el usuario decide terminar la grabación, invoca a la clase ControladorVisorDelProceso.finalizarGrabacion.

Función CancelarGrabacion_Callback

Esta función es llamada cuando el usuario decide cancelar la grabación, invoca a la clase ControladorVisorDelProceso.cancelarGrabacion.

Función inicializarLimitesEnBarrasDeControl

Esta función setea los límites en los deslizadores que permiten cambiar los valores en los parámetros de control.

Función refrescarValoresConfiguracionControl

Actualiza los valores de configuración de control en los objetos gráficos, tomandolos de la clase SetDeControlMatlab o SetDeControlManual según corresponda al modo de conexión actual. De esta manera los controles gráficos se mantienen sincronizados con los valores en los objetos que influyen en el control y grabación de los datos del proceso.

Función GuardarConfigDeControl_Callback

Esta función almacena los parámetros de control con el valor actual en un archivo, cuando el usuario así lo decida mediante las opciones de menú.

Función btnDesconectar_Callback

Esta función responde al seleccionar desconectar o bien cerrar la ventana.

Función axesVisorProceso_ButtonDownFcn

Esta función se ejecuta al hacer click sobre el gráfico del proceso y abre la ventana del gráfico ampliado, es decir GraficoDelProcesoGUI.

Función btnAgregarComentario_Callback

Esta función responde al evento de crear un nuevo comentario mientras se está grabando.

Pantalla GraficoDelProcesoGUI

Esta pantalla se abre cuando se desea ampliar el gráfico del proceso existente en la pantalla VisorDelProcesoGUI haciendo click sobre él. Esta pantalla muestra el gráfico completo con la posibilidad de hacer zoom en un área, lo que no es posible en el visor del proceso.
Esta misma pantalla es reutilizada cuando se abre un proceso grabado completo, en cuyo caso no aparece la pantalla anterior, dado que no hay variación en el tiempo.

Función GraficoDelProcesoGUI_OpeningFcn

Al igual que en las pantallas anteriores, esta función se ejecuta al abrir inicialmente la ventana. Tiene el código de inicialización de la misma. A continuación se muestra la firma de la misma.

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

Función toggleZoom_Callback

Esta función se ejecuta cuando cambia el estado del botón de opción de zoom.

Función toggleGHorizontal_Callback

Esta función se ejecuta cuando se activa o desactiva la grilla horizontal.

Función toggleGVertical_Callback

Esta función se ejecuta cuando se activa o desactiva la grilla vertical.

Función graficarProceso

Esta función obtiene la información del proceso utilizando la clase Proceso, muestras y comentarios para graficar el proceso en los ejes de la pantalla.

Función graficarComentario

Ubica el comentario en el gráfico, es llamada por la función graficarProceso para cada comentario agregado.

Otras Funciones

Función agregarMuestra

Esta función no pertenece a ninguna clase en particular, pero está dentro del paquete GraficoProceso. Es llamada por el timer que adquiere los datos de la placa adquisidora, para actualizar el gráfico existente en la pantalla VisorDelProcesoGUI.

Función agregarMuestraGrafico

Esta función no pertenece a ninguna clase al igual que la anterior y su función es similar, sólo que actualiza el gráfico en la pantalla ampliada del proceso, es decir GraficoDelProcesoGUI, siempre que no se esté en modo zoom.