DISEÑO DE UNA INTERFAZ GRAFICA DE USUARIO EN MATLAB PARA LA SIMULACION DE UN GPC (GENERALIZED PREDICTIVE CONTROL)

CARLOS ENRIQUE CALDERON

UNIVERSIDAD SURCOLOMBIANA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
NEIVA
2007

DISEÑO DE UNA INTERFAZ GRAFICA DE USUARIO EN MATLAB PARA LA SIMULACION DE UN GPC (GENERALIZED PREDICTIVE CONTROL)

CARLOS ENRIQUE CALDERON

Trabajo de grado presentado como requisito para optar al título de Ingeniero Electrónico

Director
AGUSTIN SOTO OTALORA
Docente Programa Ingeniería Electrónica

UNIVERSIDAD SURCOLOMBIANA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
NEIVA
2007

Nota de aceptación

_____

_____

_____

_____

_____
Firma del presidente del jurado

_____
Firma del jurado

_____
Firma del jurado

Neiva, 21 de agosto de 2007

A mi padre Isauro, mi madre Gloria, mis hermanos, mi hijo Juan Sebastián y esposa Sandra, que con su trabajo y dedicación familiar, hicieron posible el cumplimiento de un sueño conjunto que hoy se está haciendo realidad.

Carlos Enrique Calderón.

AGRADECIMIENTOS

El autor expresa sus agradecimientos:

A Dios

Al ingeniero Agustín Soto Otalora. Docente de la Universidad Surcolombiana y Director del proyecto de grado.

A los ingenieros Ricardo Troncoso y Oscar Barrero por su colaboración y orientación en la etapa de estudio.

A mis padres y esposa, que me han apoyado incondicionalmente y a todos aquellos que de una u otra forma, me colaboraron en la realización de éste proyecto.

CONTENIDO

pág.

LISTA DE FIGURAS

pág.

LISTA DE ANEXOS

pág.

RESUMEN

El control predictivo basado en modelo (Model Predictive Control, MPC) se ha desarrollado considerablemente en las últimas décadas tanto en la industria como en la comunidad de investigación. Este éxito se debe a que el Control Predictivo basado en Modelo es quizás la forma más general de formular el problema de control en el dominio del tiempo. El control predictivo integra control optimo, control estocástico, control de procesos con tiempos muertos, procesos multivariables y utiliza las referencias futuras cuando están disponibles. Al utilizar una estrategia con horizonte de control finito permite la consideración de restricciones y procesos no lineales.

Además, se presentará cómo implementar el diseño de un controlador predictivo mediante el entorno de desarrollo de interfaz grafico de usuario de MATLAB (GUIDE) donde el usuario ingresa la planta en tiempo discreto y las condiciones de diseño, para obtener el controlador.

ABSTRACT

The model predictive control (Model Predictive Control, MPC) has been developed considerably in the last decades as much in the industry as in the investigation community. This success is due to that the Model Predictive Control is maybe the most general form of formulating the control problem in the domain of the time. The predictive control integrates optimal control, stochastic control, control of processes with time outs, processes multivariables and it uses the future references when they are available. When using a strategy with horizon of finite control it allows the consideration of restrictions and non lineal processes.

Also, It'll be presented how to implement the design of controller predictive through the MATLAB graphical user interface development environment (GUIDE) where the user enters the plant in discrete time and the design conditions, to obtain the controller .

INTRODUCCION GENERAL

El control predictivo tiene como objetivo resolver de forma efectiva, problemas de control y automatización de procesos industriales que se caractericen por presentar un comportamiento dinámico complicado, multivariable, y/o inestable. La estrategia de control en que se basa este tipo de control, utiliza el modelo matemático del proceso a controlar para predecir el comportamiento futuro de dicho sistema, y en base a este comportamiento futuro puede predecir la señal de control futura.

El control predictivo integra disciplinas como el control óptimo, control estocástico, control de procesos con retardo de tiempo, control multivariable, control con restricciones.

El tipo de control predictivo a utilizar en la presente tesis, es el Control Predictivo Basado en Modelo (CPBM), conocido también como Model Based Predictive Control (MBPC) o simplemente Model Predictive Control (MPC). Esta estrategia también se conoce como control por horizonte deslizante, por ser ésta la forma en la que se aplican las señales de actuación. Existen muchos algoritmos de control predictivo que han sido aplicados con éxito: GPC, IDCOM, DMC, APC, PFC, EPSAC, RCA, MUSMAR, NPC, UPC, SCAP, HPC, etc.

El control predictivo basado en modelo se puede definir como una estrategia de control que se basa en la utilización de forma explícita de un modelo matemático interno del proceso a controlar (modelo de predicción), el cual se utiliza para predecir la evolución de las variables a controlar a lo largo de un horizonte temporal de predicción especificado por el operador, de este modo se puede calcular las variables manipuladas futuras (señal de control futura) para lograr que en dicho horizonte, las variables controladas converjan en sus respectivos valores de referencia.

El MPC se enmarca dentro de los controladores óptimos, es decir, aquellos en los que las actuaciones responden a la optimización de un criterio. El criterio a optimizar, o función de costo, está relacionado con el comportamiento futuro del sistema, que se predice gracias a un modelo dinámico del mismo, denominado modelo de predicción. El intervalo de tiempo futuro que se considera en la optimización se denomina horizonte de predicción. Dado que el comportamiento futuro del sistema depende de las actuaciones que se aplican a lo largo del horizonte de predicción, son éstas las variables de decisión respecto a las que se optimiza el sistema. La aplicación de estas actuaciones sobre el sistema conduce a un control en bucle abierto.

La posible discrepancia entre el comportamiento predictivo y el comportamiento real del sistema crean la necesidad de imponer cierta robustez al sistema incorporando realimentación del mismo. Esta realimentación se consigue gracias a la técnica del horizonte deslizante que consiste en aplicar las actuaciones obtenidas durante un periodo de tiempo, tras el cual se muestrea el estado del sistema y se resuelve un nuevo problema de optimización. De esta manera, el horizonte de predicción se va deslizando a lo largo del tiempo.

Una de las propiedades más atractivas del MPC es su formulación abierta, que permite la incorporación de distintos tipos de modelos de predicción, sean lineales o no lineales, monovariables o multivariables, y la consideración de restricciones sobre las señales del sistema. Esto hace que sea una estrategia muy utilizada en diversas áreas del control. El CPBM es una de las pocas técnicas que permiten controlar sistemas con restricciones incorporando éstas en el propio diseño del controlador.

Estas características han hecho del control predictivo una de las escasas estrategias de control avanzado con un impacto importante en problemas de ámbito industrial Por tal motivo es importante resaltar que el control predictivo se ha desarrollado en el mundo de la industria, y ha sido la comunidad investigadora la que se ha esforzado en dar un soporte teórico a los resultados prácticos obtenidos.

Merece la pena destacar que el control predictivo es una técnica muy potente que permite formular controladores para sistemas complejos y con restricciones. Esta potencia tiene un precio asociado: el coste computacional y la sintonización del controlador.

OBJETIVOS

General

- Desarrollar una interfaz grafica de usuario en Matlab para la implementación de un simulador de control predictivo generalizado.

Específicos

- Conocer la estructura y los elementos básicos de los controladores predictivos.
- Aprender a utilizar las funciones básicas que tiene Matlab para el desarrollo de interfaces graficas de usuario.

1. TEORIA DE CONTROL PREDICTIVO[1]

1.1 CONCEPTOS BÁSICOS DE CONTROL PREDICTIVO

El Control Predictivo Basado en Modelo, Model Based Predictive Control (MBPC o MPC) constituye un campo muy amplio de métodos de control e involucra diversas disciplinas como son: control óptimo, control estocástico, control de procesos con tiempos muertos, control multivariable o control con restricciones.

El Control Predictivo abarca un campo muy amplio de métodos de control desarrollados en torno a ciertas ideas comunes. Estos métodos de diseño permiten desarrollar controladores lineales que poseen prácticamente la misma estructura y presentan suficientes grados de libertad.

Las ideas que representan esta familia de controladores predictivos son básicamente:

- Uso explicito de un modelo para predecir la salida del proceso en futuros instantes de tiempo (horizonte).
- Calculo de las señales de control minimizando una función de costo.
- Estrategia deslizante, de forma que en cada instante de tiempo el horizonte se va desplazando hacia el futuro, lo que implica aplicar la primera señal de control en cada instante y desechar el resto, repitiendo el cálculo en cada instante de muestreo.

Dependiendo del modelo usado para representar el proceso, los ruidos y de la función de costo a minimizar, se pueden emplear diferentes algoritmos de MPC.

Aunque las diferencias puedan parecer pequeñas *a priori*, pueden provocar distintos comportamientos en lazo cerrado, siendo críticas para el éxito de un determinado algoritmo en una determinada aplicación.

El Control Predictivo ha encontrado una gran aceptación tanto en aplicaciones industriales como en el mundo académico. Actualmente existen numerosas aplicaciones de controladores predictivos funcionando con éxito, tanto en la industria de procesos como en control de motores o Robótica. El buen funcionamiento de estas aplicaciones muestra el potencial que tiene el MPC para conseguir sistemas de control de elevadas prestaciones capaces de operar sin apenas intervención durante largos periodos de tiempo.

[1] http://www.esi2.us.es/~bordons/apuntes_MPC.pdf

Entre las ventajas que podemos destacar del MPC sobre otros métodos, sobresalen:

- Resulta especialmente atractivo para personas sin un conocimiento profundo de control, puesto que los conceptos resultan muy intuitivos, a la vez que la sintonización es relativamente fácil.

- Puede ser usado para controlar una gran variedad de procesos, desde procesos con dinámicas relativamente simples hasta otros mas complejos incluyendo sistemas con grandes retardos, de fase no mínima o inestables.

- Permite tratar con facilidad el caso multivariable.

- Posee intrínsecamente compensación del retardo.

- Se pueden incluir restricciones, las cuales son introducidas de forma sistemática durante el proceso de diseño.

- Es muy útil cuando se conocen las referencias futuras (robótica o procesos en *batch*).

- Es una metodología completamente abierta basada en algunos principios básicos que permite futuras extensiones.

Uno de los inconvenientes que presenta, es la carga de cálculo necesaria para la resolución de algunos algoritmos. Pero posiblemente el mayor inconveniente venga marcado por la necesidad de disponer de un modelo apropiado del proceso.

El algoritmo de diseño esta basado en el conocimiento previo del modelo y es independiente de este, pero resulta evidente que las prestaciones obtenidas dependerán de las diferencias existentes entre el proceso real y el modelo usado.

1.2 ESTRATEGIA DEL CONTROL PREDICTIVO

La metodología de todos los controladores predictivos se caracteriza por la siguiente estrategia, representada en la figura 1:

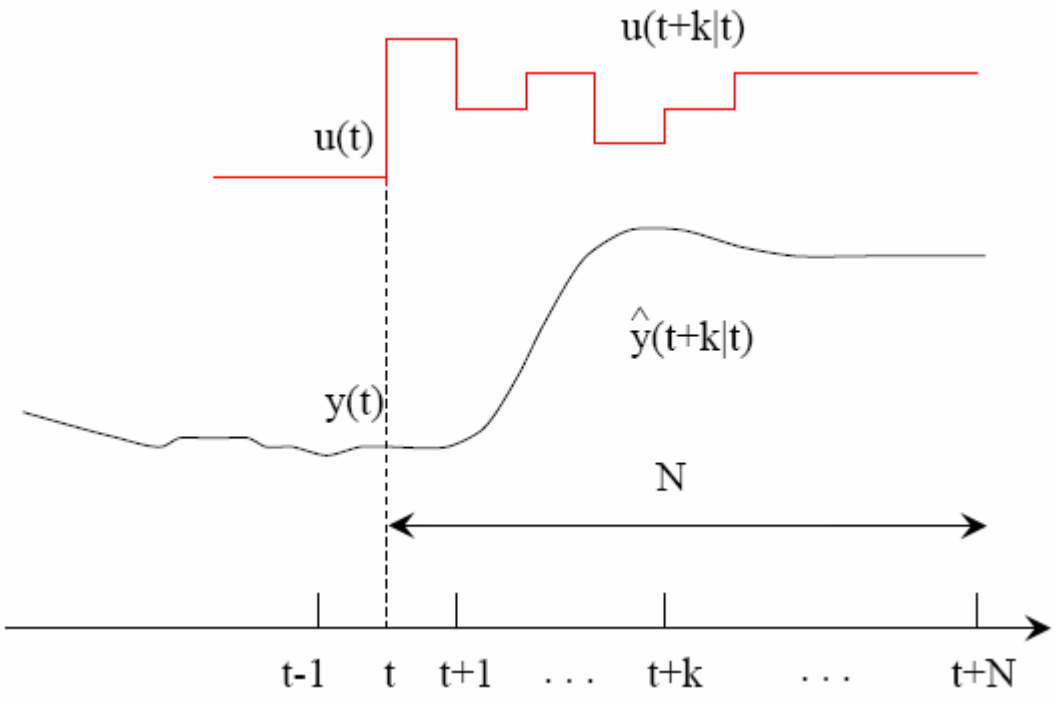

Figura 1. Estrategia MPC.

1. Las salidas futuras para un horizonte determinado N, llamado horizonte de predicción, se predicen cada instante t utilizando el modelo del proceso. Estas predicciones de la salida $y(t+k\,|t)$ para $k=1...N$ dependen de los valores conocidos hasta el instante t (entradas y salidas conocidas) y de las señales de control $u(t+k\,|t),\ k=0...N-1$, que deben ser calculadas y enviadas al sistema.

2. La secuencia de señales de control futuras se calcula minimizando un criterio para mantener al proceso lo más cerca posible de la trayectoria de referencia $w(t+k)$. Este criterio toma normalmente la forma de una función cuadrática del error entre la salida predicha y la trayectoria de referencias futuras. En la mayor parte de los casos se incluye también el esfuerzo de control dentro de la función objetivo. La solución explicita se puede obtener cuando el criterio es cuadrático y el modelo lineal; en caso contrario se ha de utilizar un método numérico para buscar la solución.

3. La señal de control $u(t\,|t)$ se envía al proceso mientras que el resto de las señales calculadas no se consideran, ya que en el instante siguiente de

muestreo $y(t+1)$ es ya conocida y los pasos anteriores se repiten con este nuevo valor. Por lo que $u(t+1 \mid t+1)$ se calcula con información diferente y en principio será también diferente de $u(t+1 \mid t)$.

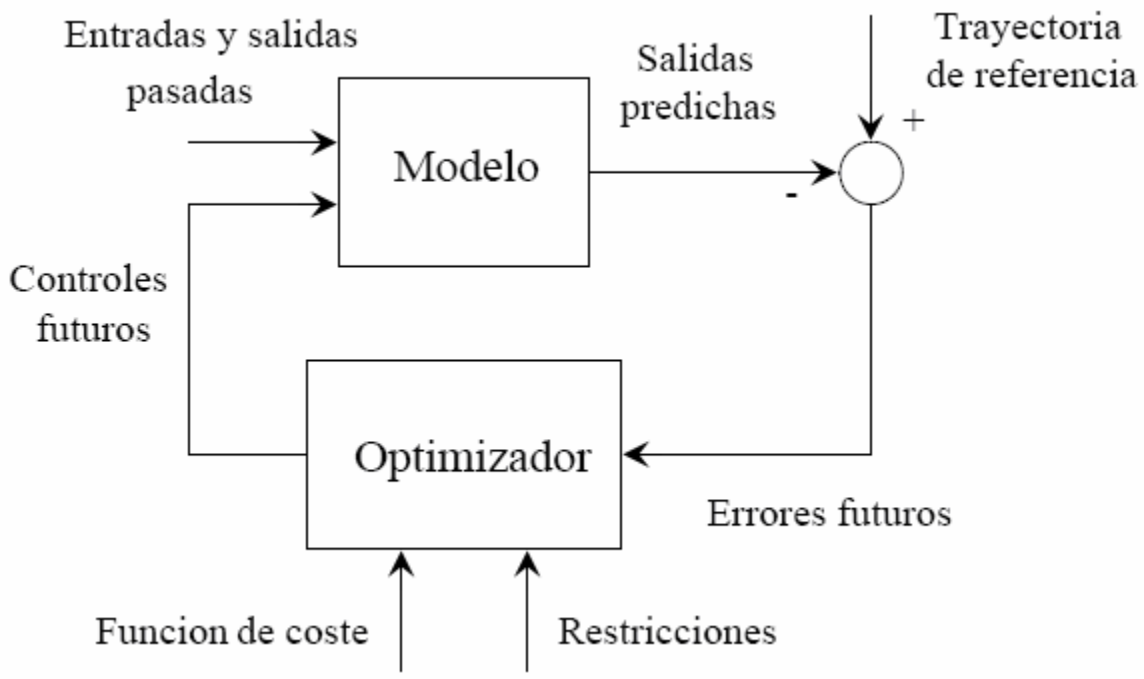

Figura 2. Estructura básica del MPC.

La figura 2 muestra la estructura básica del control predictivo. Se usa un modelo para predecir la evolución de la salida o estado del proceso a partir de las señales de entrada y salidas conocidas. Las acciones de control futuras se calculan con el optimizador, que considera la función de costo y las posibles restricciones.

El modelo del proceso es de gran importancia en el diseño del controlador. Además este modelo debe ser capaz de capturar la dinámica del proceso para predecir de forma precisa la evolución del sistema. Al mismo tiempo, debe ser muy simple de implementar y de entender.

El tipo de modelo utilizado difiere fundamentalmente de las distintas metodologías del control predictivo.

Otra parte fundamental de la estructura es el optimizador ya que permite obtener las acciones de control a aplicar. Si la función de costo es cuadrática, el modelo lineal y no existen restricciones, se puede obtener una solución explicita. Si este no es el caso se debe recurrir a un algoritmo numérico de optimización que

requiere mayor capacidad de cálculo. El tamaño del problema resultante depende del número de variables, de los horizontes de control y predicción y del número de restricciones, aunque se puede decir que en general problemas de optimización resultantes en este contexto son problemas más bien modestos.

Como podemos observar la estrategia de control predictivo es muy similar a la estrategia que se utiliza cuando se conduce un automóvil. El conductor conoce la trayectoria de referencia deseada para un horizonte de control finito. Tomando en consideración las características del automóvil (modelo mental del automóvil) decide que acción de control tomar (acelerador, frenos, volante, marchas) para seguir la trayectoria deseada. Solo la primera acción de control de la secuencia calculada mentalmente es aplicada por el conductor en cada instante y el procedimiento se repite en los sucesivos instantes utilizando el concepto de horizonte deslizante.

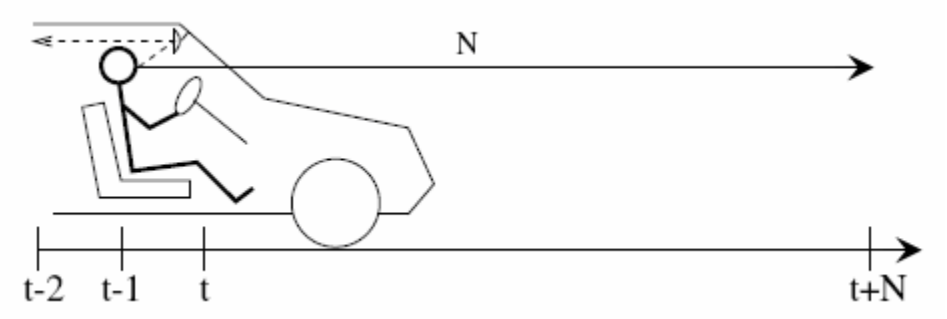

Figura 3. Analogía MPC.

Como ya sabemos cuando se utiliza un esquema de control clásico como PID se utilizan solo las señales pasadas. Esta forma de conducir el automóvil seria como conducir utilizando el espejo retrovisor tal como se muestra en la figura 3. Esta analogía no es totalmente justa con los PIDs, porque el control predictivo utiliza más información (trayectoria de referencia). Nótese que si se le proporciona al PID como referencia un punto en la trayectoria futura la diferencia entre ambas estrategias de control no parecería tan abismal.

2. FUNDAMENTOS DE INTERFACES GRAFICAS DE USUARIO CON MATLAB

MATLAB permite desarrollar de manera simple un conjunto de pantallas con botones, menús, ventanas, etc., que permiten utilizar fácilmente programas realizados en el entorno ***Windows***. Este conjunto de herramientas se denomina ***interfaz de usuario.***

Para ello disponemos del módulo llamado GUIDE (***Graphical User Interface Development Environment***).

Para poder hacer algún programa que utilice las capacidades gráficas avanzadas de MATLAB hay que conocer algunos conceptos previamente.

Los gráficos de MATLAB tienen una estructura jerárquica formada por ***objetos*** de distintos ***tipos***. Esta jerarquía tiene forma de *árbol,* con ***objetos padres*** e ***hijos***. Por ejemplo, todos los objetos ***ventana*** son hijos de ***pantalla***, y cada ***ventana*** es padre de los objetos ***ejes***, ***controles*** o ***menús*** que están por debajo. A su vez los elementos gráficos (líneas, polígonos, etc.) son hijos de un objeto ***ejes***, y no tienen otros objetos que sean sus hijos.

Cuando se borra un objeto de MATLAB automáticamente se borran todos los objetos que son sus descendientes. Por ejemplo, al borrar unos ejes, se borran todas las líneas y polígonos que son hijos suyos.

2.1 IDENTIFICADORES DE OBJETOS EN MATLAB

Cada uno de los objetos de MATLAB tiene un ***identificador único*** (***handle***). A los identificadores se les llama ***handle*** o ***id***, indistintamente. Algunos gráficos tienen muchos objetos, en cuyo caso tienen múltiples *handles*. El ***objeto raíz*** (pantalla) es siempre único y su identificador es el ***cero***. El identificador de las ventanas es un entero, que aparece en la barra de nombre de dicha ventana. Los identificadores de otros elementos gráficos son números *float*, que pueden ser obtenidos como valor de retorno y almacenados en variables de MATLAB.

MATLAB puede tener varias ventanas abiertas, pero siempre hay una y sólo una que es la ***ventana activa***. A su vez una ventana puede tener varios *ejes* (*axes*), pero sólo unos son los ***ejes activos***. MATLAB dibuja en los ejes activos de la ventana activa. Los identificadores de la ventana activa, de los ejes activos y del objeto activo se pueden obtener respectivamente con los comandos ***gcf*** (*get current figure*), ***gca*** (*get current axes*) y ***gco*** (*get current object*):

- **gcf** devuelve un entero, que es el ***handle*** de la ventana activa

- **gca** devuelve el ***handle*** de los ejes activos
- **gco** devuelve el ***handle*** del objeto activo

Los objetos se pueden borrar con el comando ***delete***:

- **delete(handle)** borra el objeto correspondiente y todos sus hijos

Los valores de retorno pueden ser almacenados en variables para un uso posterior.

2.2 CONSTRUCCIÓN DE INTERFACES DE USUARIO (GUIDE)

MATLAB, a partir de la versión 5.0, ha incorporado un módulo llamado **GUIDE** (Graphical User Interface Development Environment) que permite crear de modo interactivo interfaces de usuario, al modo de Visual Basic.

GUIDE se ejecuta a partir de la línea de comandos de MATLAB, tecleando:
>>guide
y pulsando ***Intro***. A continuación se abre la ventana ***Guide Control Panel*** (***GCP***), mostrada a continuación:

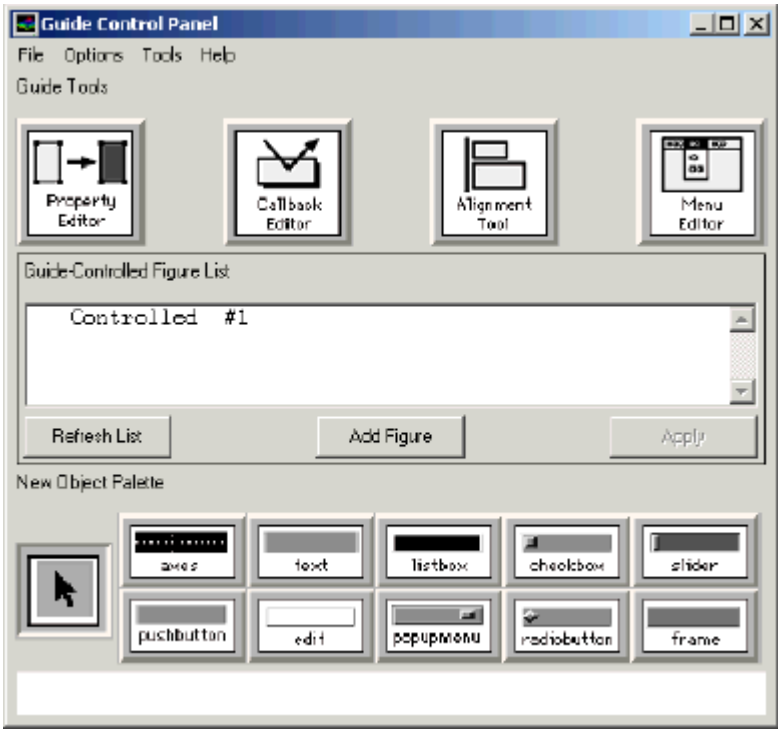

Figura 4. Guide control panel

Se abre también una figura en blanco, sobre la que el diseñador deberá ir situando los distintos controles con el ratón, hasta terminar con el aspecto requerido. Dicha ventana esta dividida en tres partes principales:

La ***parte superior*** contiene cuatro grandes botones o iconos, correspondientes a los otros cuatro grandes módulos de GUIDE. De izquierda a derecha aparecen los iconos correspondientes a:

- el *Editor de Propiedades* (***Property Editor***),
- el *Editor de Llamadas* (***Callback Editor***),
- el *Editor de Alineamientos* (***Alignement Editor***)
- el *Editor de Menús* (***Menu Editor***)

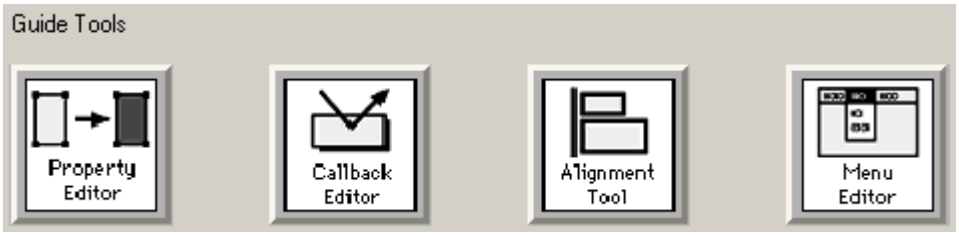

Figura 5. Guide Tools

Estos editores se pueden hacer visibles desde la línea de comandos de MATLAB (***propedit***, ***cbedit***, ***align*** y ***menuedit***), clicando en dichos iconos o seleccionándolos en el menú ***Tools*** de la ventana en la que se está haciendo el diseño.

La ***parte central*** de la ***GCP*** contiene la lista de ventanas o figuras de MATLAB controladas por GUIDE.

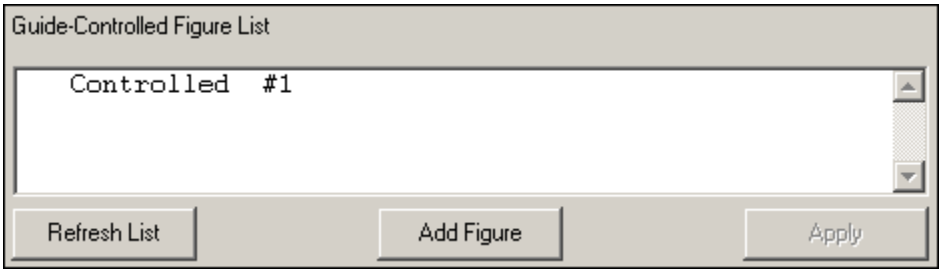

Figura 6. Parte central del guide control panel

En este caso sólo hay una, pero podría haber varias. Cada una de las figuras puede estar dos estados: *controlada* (***Controlled***) y *activa* (***Active***). Estos dos estados se corresponden con los modos "de diseño" y "de ejecución" de otras aplicaciones. Para que una figura pase de ***Controlled*** a ***Active*** hay que realizar dos acciones: 1.- Clicar sobre la línea correspondiente en la lista de figuras, con lo cual el mensaje en dicha lista pasa de un estado a otro, y, 2.-Clicar sobre el botón ***Apply***, con lo cual el estado seleccionado pasa a ser el estado real de la figura. Junto al botón ***Apply*** aparece otro botón llamado ***Add Figure*** que permite crear una nueva figura cuando se desee

La ***parte inferior*** de la ***GCP*** contiene iconos correspondientes a los elementos de interface de usuario (o controles) soportados por MATLAB, que son los siguientes (de izquierda a derecha y de arriba a abajo): ***axes***, ***text***, ***listbox***, ***checkbox***, ***slider***, ***pushbutton***, ***edit***, ***popupmenu***, ***radiobutton*** y ***frame***.

Figura 7. Parte inferior del guide control panel

Para crear uno de estos controles basta con dar clic sobre el icono correspondiente y luego ir a la figura en que se desee introducirlo (que deberá estar en estado ***Controlled***), clicar y arrastrar con el botón izquierdo del ratón pulsado para dar al control la posición y tamaño deseado. El nuevo control puede desplazarse y cambiarse de tamaño con ayuda del ratón, al igual que la propia ventana en la que ha sido situado Se puede observar que falta el icono correspondiente a ***togglebutton***. Para crear un botón de este tipo se crea un ***pushbutton*** y se cambia la propiedad ***Style*** a ***togglebutton***

2.2.1 El Editor de Propiedades (Property Editor). En la parte superior del ***PE*** aparece una lista jerarquizada de los controles presentes en la figura (***Object Browser***). En este caso todos los controles son "hijos" de la figura nº 1, lo cual se muestra en el hecho de que todos aparezcan algo desplazados hacia la derecha (en este caso, en la figura solo hay una barra de desplazamiento o slider, y un cuadro de texto). Para cada elemento se muestra el tipo (propiedad ***Style***) y un nombre (***Slider1*** para el elemento seleccionado). Este nombre se define por medio de la propiedad ***Tag***. Los nombres utilizados en este caso son los nombres por defecto propuestos por GUIDE.

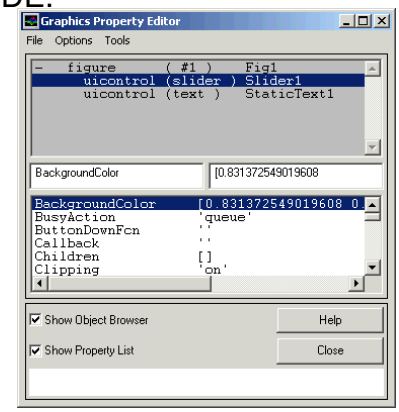

Figura 8. El editor de propiedades.

En la parte central del ***PE* (Property Editor)** aparece una lista o relación de propiedades del objeto seleccionado en la parte superior. Las propiedades no se pueden editar directamente sobre esta lista: hay que seleccionar una propiedad y darle valor en la caja de texto que aparece inmediatamente encima de la lista, a la derecha (a la izquierda aparece el nombre de la propiedad). Para las propiedades que sólo pueden tomar ciertos valores, aparece una lista desplegable que los muestra y ayuda a elegirlos. Para seleccionar una propiedad basta teclear sus primeras letras en la caja de texto que contiene el nombre y pulsar ***Intro***.

Los controles ***listbox*** y ***popupmenu*** contienen un conjunto de opciones. El texto de esas opciones se puede almacenar en un ***cell array*** del espacio de trabajo base y luego la variable se introduce en la propiedad ***String*** del control (crea una copia llamada ***towork***).

Para las barras de desplazamiento (***sliders***) los valores máximos y mínimos vienen dados por las propiedades ***max*** y ***min***. Los incrementos pequeño y grande vienen definidos por la propiedad ***SliderStep***, que es un vector de dos elementos.

2.2.2 El Editor de Llamadas (Callback Editor). El ***Callback Editor (CE)*** es uno de los componentes más importantes de GUIDE, porque permite definir la forma en la que los distintos controles responden a las acciones del usuario (***eventos***).
La respuesta que se desea obtener cuando el usuario realiza una determinada acción sobre un control se obtiene programando los ***callbacks***, que definen el código que se debe ejecutar en ese caso.

Este código se asocia con el control por medio del ***CE***, en cuya parte superior vemos una lista de objetos (***Object Browser***). Seleccionando un objeto, en la caja de texto editable que aparece en el centro puede escribirse el código que se desea ejecutar al producirse el evento correspondiente. El evento se puede seleccionar en la lista desplegable que hay debajo del ***Object Browser***.

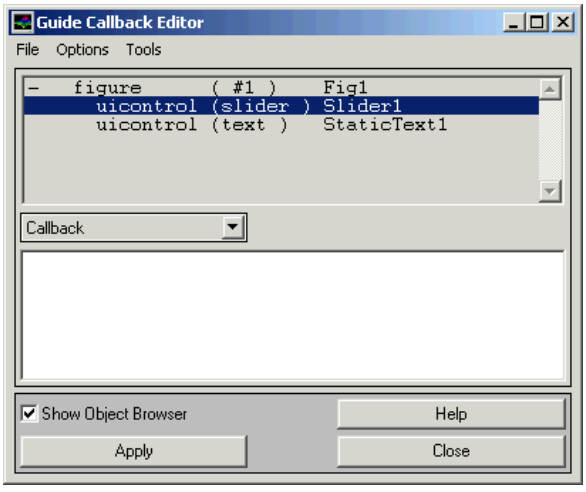

Figura 9. El editor de llamadas

El ***CE* (Callback Editor)** permite definir ***callbacks*** correspondientes a eventos más especializados (Los mostrados en el menú desplegable).

2.2.3 El Editor de Alineamientos (Alignment Editor). El ***Alignment Editor (AE)*** es una herramienta auxiliar que permite que los controles situados en una ventana por medio de GUIDE, aparezcan uniformemente alineados o distribuidos. Su funcionamiento es bastante fácil.

En la parte superior se muestra de nuevo la lista de objetos (***Object Browser***), en la que deberán estar seleccionados los objetos que se desea alinear o distribuir (para seleccionar varios objetos basta clicar sucesivamente sobre ellos manteniendo pulsada la tecla de ***Mayúsculas***). El ***AE* (Alignment Editor)** considera una caja "imaginaria" que comprende los objetos a ser alineados o distribuidos, y que realiza su operación de alineamiento o distribución dentro de esa caja.

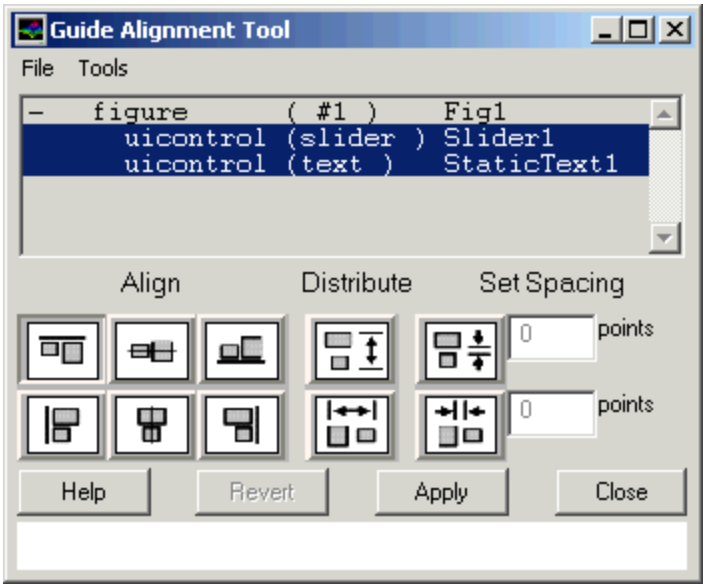

Figura 10. El editor de alineamientos

2.2.4 El Editor de Menús (Menu Editor). En toda ventana de MATLAB (***figure***) aparecen por defecto cuatro menús (***File***, ***Edit***, ***Windows*** y ***Help***). Los menús creados con el ***Menu Editor*** son menús adicionales. MATLAB permite hasta 4 niveles de menús. En la parte superior del ***Menu Editor*** aparece una representación jerárquica de los menús. Para este ejemplo se ha introducido un menú ***Color***, con dos opciones, rojo y verde.

Para introducir un nuevo menú en la jerarquía basta pulsar el botón ***New Menu*** y establecer los datos que se deseen, situándolo con los botones en la posición deseada del menú deseado.

Pulsando ***Apply*** los datos de las cajas de texto se introducen en la jerarquía visible en la parte superior de la ventana.

A la izquierda de la lista de menús aparecen cuatro botones que permiten mover los menús por la jerarquía, aumentando o reduciendo su nivel en dicha jerarquía (*promote* y *demote*), cambiando su orden o pasándolos de un menú a otro.

Debajo de la jerarquía de menús del ***ME* (Menu Editor)** se muestran existen tres cajas de texto que permiten fijar el ***Label*** (lo que aparece como título del menú), el ***Tag*** (el nombre interno del menú) y el ***callback*** (la función o los comandos que se ejecutarán al elegir ese menú).

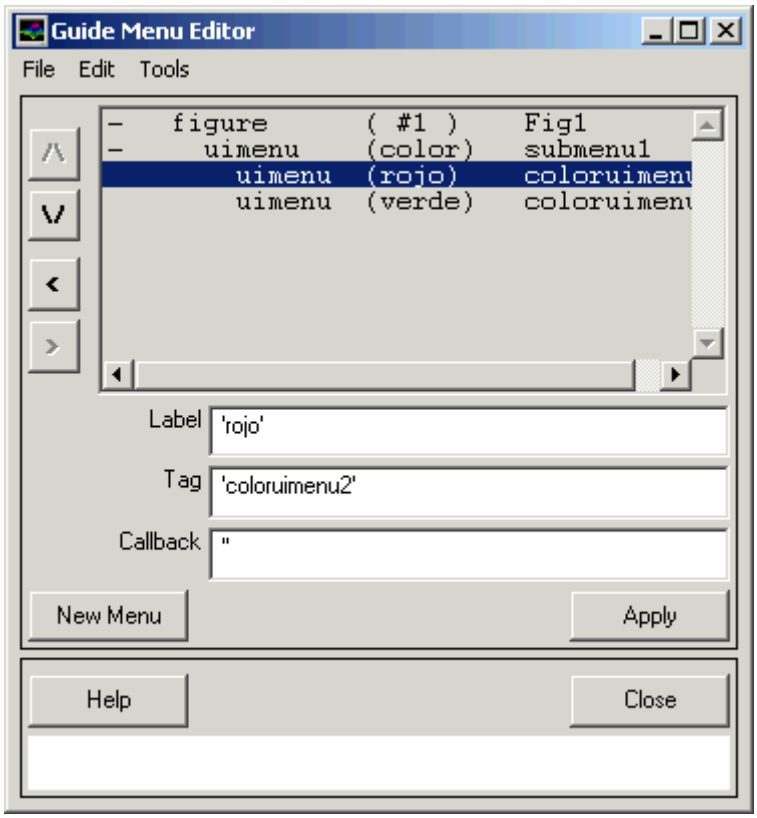

Figura 11. El editor de menus

3. IMPLETANCION EN GUI PARA DESARROLLAR EL CONTROLADOR PREDICTIVO

3.1 Primero. Modelo del Proceso

Para dar inicio se debe comenzar por tener el modelo del sistema, este como se dijo anteriormente debe ser muy fiel ya que la predicción de la secuencia de control y el éxito del control mismo dependen en su mayoría de este primer paso.

Dependiendo del algoritmo que se utilice para desarrollar el controlador, así mismo deberá ser el tipo de modelo que se use para predecir la evolución futura de las variables controladas de la planta y la función objetivo, la cual debe ser optimizada para obtener la mejor señal de control.

Los tipos de modelos que se utilizan para representar el sistema son:

- Modelo de función de transferencia
- Modelo de convolución
- Modelo de espacio de estados.

El modelo del sistema que utilizaremos para este caso será el de función de transferencia.

Para el control predictivo se debe tener la función de transferencia discretizada en la forma de modelo CARIMA (Controller Auto-Regressive Integrated Moving-Average).

En caso de no ser así, se debe discretizar encontrando un tiempo de muestreo apropiado para que el modelo siga siendo fiel al sistema real, generalmente se utiliza el teorema de Nyquist para definir dicho tiempo.

Ahora se tiene la función de transferencia discretizada

$$A(z^{-1})y(t) = B(z^{-1})z^{-d}u(t-1) + C(z^{-1})\frac{e(t)}{\Delta}$$

En donde:

- $\Delta = 1 - z^{-1}$ operador de desplazamiento hacia atrás.
- $u(t)$ y $y(t)$ son las variables de entrada y salida del sistema
- $e(t)$ es un ruido blanco.

- A, B y C son polinomios en el operador de desplazamiento hacia atrás z^{-1}
- d es el tiempo muerto del proceso o tiempo de retardo, es decir, el tiempo que tarda el sistema en responder ante un a entrada.

Para el desarrollo de este controlador el modelo la planta no presenta perturbaciones, por lo tanto el término $C(z^{-1})\frac{e(t)}{\Delta}$ vale cero. Entonces el modelo del sistema puede expresarse de la forma $G = B/A$:

$$A(z^{-1})y(t) = B(z^{-1})u(t)$$

$$G(z^{-1}) = \frac{y(t)}{u(t)} = \frac{B(z^{-1})}{A(z^{-1})}$$

Con

$$A(z^{-1}) = 1 + a_1 z^{-1} + a_2 z^{-2} + ... + a_{na} z^{-na}$$

$$B(z^{-1}) = b_1 z^{-1} + b_2 z^{-2} + ... + b_{nb} z^{-nb}$$

En matlab se introduciría el modelo que se desarrollara en la implementación del GUI de la siguiente manera

```
d=0;          % retardo del sistema
Ts=1;         % Tiempo de Muestreo (seg)
A=[1 -0.8];   % A(z^-1)
B=[zeros(1,d) 0.4 0.6];% B(z^-1)
Gz=tf(B,A,Ts,'Variable','z^-1')% G(z^-1)
```

Nuestra función de transferencia queda expresada entonces de la siguiente manera:

$$G(z^{-1}) = \frac{y(t)}{u(t)} = \frac{0.4 + 0.6z^{-1}}{1 - 0.8z^{-1}}$$

Para el desarrollo de la interfaz grafica de usuario partimos del hecho de que debemos introducir como primera medida la función de transferencia del sistema en la forma discreta, sin embargo, el programa tiene una opción la cual permite escoger la forma de introducir la función de transferencia, es decir, se puede cambiar entre continuo o discreto. Esta opción es de gran ayuda ya que si el

usuario no tiene la función de transferencia en discreto, tendría que realizar estos cálculos por aparte, lo que seria muy tedioso.

La interfaz grafica que nos permite introducir estos valores es la siguiente:

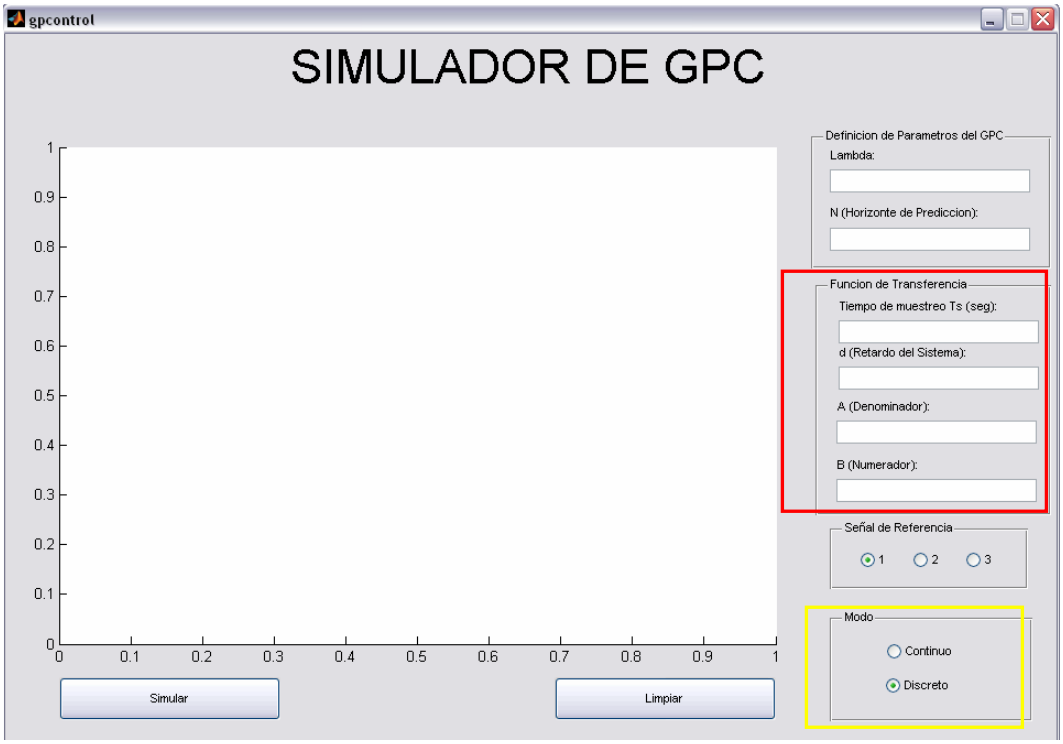

Figura 12. Simulador GPC (Función de Transferencia)

En el cuadro que aparece encerrado se encuentran las casillas en las cuales debemos introducir el denominador y el numerador en forma de vector, y adicionalmente el tiempo de muestreo, el tiempo muerto que hemos definido previamente. También tiene una opción la cual permite escoger la forma de introducir la función de transferencia, es decir, se puede cambiar entre continuo o discreto. Esta opción es de gran ayuda ya que si el usuario no tiene la función de transferencia en discreto, tendría que realizar estos cálculos por aparte, lo que seria muy tedioso.

3.2 Segundo. **Definición de la trayectoria de referencia**

El programa tiene la opción de escoger tres tipos de trayectorias de referencia diferentes, que se generan mediante el siguiente código en Matlab:

```
%%%%%  Trayectorias de Referencia  %%%%%%%%

switch get(hObject,'Tag')
    case 'radiobutton3'
        handles.r = ones(40,1);       % Señal Paso
    case 'radiobutton4'
```

```
        handles.r = sin(0.25*(1:40))'; % Señal Seno
    case 'radiobutton5'
        handles.r =sawtooth(0.66*(0:40))'; % Señal Diente de Sierra
end
```

3.3 Tercero. **Definición de la Función objetivo.**

El Control Predictivo Generalizado usa una función de costo cuadrática de la forma:

$$J(N_1,N_2,N_u)=\sum_{j=N_1}^{N_2}\delta(j)\left[\hat{y}(t+j|t)-w(t+j)\right]^2+\sum_{j=1}^{N_u}\lambda(j)\left[\Delta u(t+j-1)\right]^2$$

En donde:

- $\delta(j)$ y $\lambda(j)$ son secuencias que ponderan el comportamiento futuro. Normalmente estos valores se consideran constantes.
- $\hat{y}(t+j|t)$ es la predicción óptima j pasos hacia delante de la salida del proceso con datos conocidos hasta el instante t.
- $w(t+j)$ es la trayectoria de referencia, la cual puede ser considerada constante e igual a la referencia actual.

El objetivo principal es minimizar esta función cuadrática, puesto que esta se encarga de medir la distancia entre la salida predicha del sistema y la trayectoria de referencia, hasta el horizonte de predicción. Esta minimización se logra calculando una secuencia de control futura $u(t),\ u(t+1),\ldots$ de tal manera que la salida futura del proceso $y(t+j)$ permanezca próxima a $w(t+j)$, que es la trayectoria de referencia.

Para nuestro caso hemos definido en Matlab tres tipos de trayectorias diferentes de la siguiente manera:

3.3.1 Parámetros de la función de costo:

- N_1 y N_2 :Horizontes mínimo y máximo de costo (o de predicción)
  Estos marcan los límites de los instantes en que se desea que la salida siga a la referencia. Así, si se toma un valor grande de N_1 es porque no importa que haya errores en los primeros instantes, lo cual provoca una respuesta suave del proceso.
- N_u el horizonte de control.

Los valores N_1, N_2 y N_u que marcan los horizontes están definidos en Matlab de la siguiente manera:

```
N1=d+1;        % Horizonte mínimo de costo
Nu=d+N;        % Horizonte máximo de costo
N=3;           % Horizonte de Predicción
```

Para nuestro caso $N_u = N_2$. Y el peso para la entrada $u(t)$ en la función de costo $\lambda(j)$ es igual a 0.8. Este valor se expresa en Matlab de la siguiente forma:

```
lambda=0.8;    % Peso para la entrada u en la función de costo
```

3.3.2 Obtención de la matriz de predicción del Controlador.

Para obtener esta matriz lo primero que debemos hacer es hallar los polinomios del predictor E_j y F_j. Esto se lleva a cabo resolviendo una ecuación diofántica expresada como:

$$1 = E_j(z^{-1})\ \tilde{\mathrm{A}} + z^{-j} F_j(z^{-1}),\ para\ j = 1..3$$

Luego los primeros términos E_1 y F_1, se pueden obtener dividiendo 1 entre $\tilde{\mathrm{A}}(z^{-1})$. En donde el cociente de la división es E_1 y el resto F_1. Los valores obtenidos son:

$$E_1(z^{-1}) = 1 \text{ y } F_1(z^{-1}) = 1.8 - 0.8z^{-1}$$

Para encontrar los demás términos E_2 y E_3, podemos utilizar la siguiente ecuación:

$$E_{j+1}(z^{-1}) = E_j(z^{-1}) + e_{j+1,j} z^{-j}$$

Con

$$e_{j+1,j} = f_{j,0}$$

Para el caso de los términos F_2 y F_3, utilizamos la siguiente expresión:

$$F_{j+1}(z^{-1}) = f_{j+1,0} + f_{j+1,1} z^{-1} + \ldots + f_{j+1,na} z^{-na}$$

Una vez desarrolladas estas expresiones obtenemos estos valores:

$$E_2(z^{-1}) = 1 + 1.8z^{-1} \text{ y } F_2(z^{-1}) = 2.44 + 1.44z^{-1}$$

$$E_3(z^{-1}) = 1 + 1.8z^{-1} + 2.44z^{-2} \text{ y } F_3(z^{-1}) = 2.952 - 1.952z^{-1}$$

Con estos valores y el polinomio $B(z^{-1}) = 0.4 + 0.6z^{-1}$, tenemos que encontrar los elementos de $G_j(z^{-1})$ que son los que van a formar parte de la matriz de predicción. El procedimiento para hacerlo es el siguiente:

$$G_1(z^{-1}) = E_1 * B(z^{-1})$$
$$G_2(z^{-1}) = E_2 * B(z^{-1})$$
$$G_3(z^{-1}) = E_3 * B(z^{-1})$$

Luego,

$$G_1(z^{-1}) = 0.4 + 0.6z^{-1}$$
$$G_2(z^{-1}) = 0.4 + 1.32z^{-1} + 1.08z^{-2}$$
$$G_3(z^{-1}) = 0.4 + 1.32z^{-1} + 2.056z^{-2} + 1.4641.08z^{-3}$$

Ahora, debemos hallar los términos que van a formar parte de la matriz de predicción:

$$G = \begin{bmatrix} g_0 & 0 & 0 \\ g_1 & g_0 & 0 \\ g_2 & g_1 & g_0 \end{bmatrix}$$

En donde

$$g_0 = 0.4$$
$$g_1 = 1.32$$
$$g_2 = 2.056$$

Finalmente tenemos que la matriz de predicción va a ser igual a:

$$G = \begin{bmatrix} 0.4 & 0 & 0 \\ 1.32 & 0.4 & 0 \\ 2.056 & 1.32 & 0.4 \end{bmatrix}$$

En matlab esto se hace de la siguiente manera:

```
%%%% Respuesta Paso %%%%%

[g,t]=step(Gz); % Se aplica una señal paso para hallar los términos gi
gk=g(1:N);      % que forman la matriz de predicción.
G=zeros(N); % Matriz de Predicción del GPC
for i=1:N
    G(i:N,i)=gk(1:N+1-i);
end
```

En el GUI desarrollado tenemos la siguiente ventana, en la cual debemos introducir los valores para el peso que tendrá la entrada en la función de costo y el horizonte de predicción.

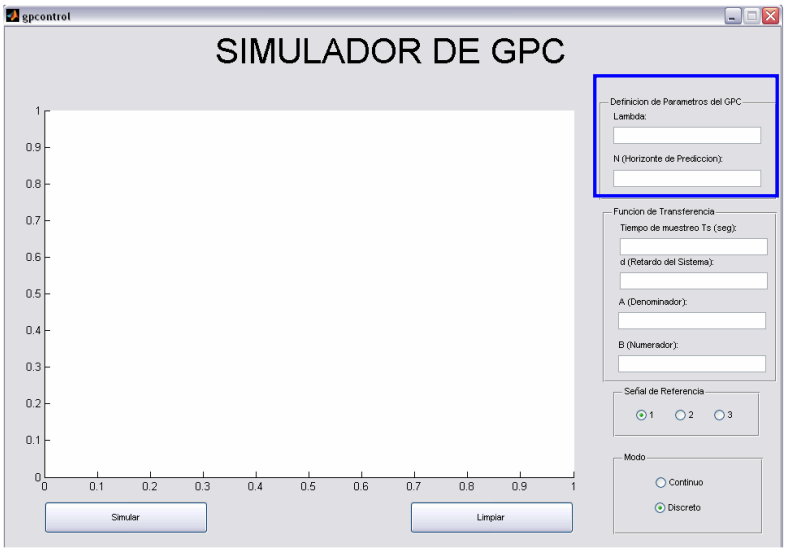

Figura 13. Simulador GPC (Parámetros del GPC)

A continuación podemos observar la casilla en la cual podemos escoger las diferentes trayectorias de referencia, por defecto la trayectoria de referencia escogida es una señal paso.

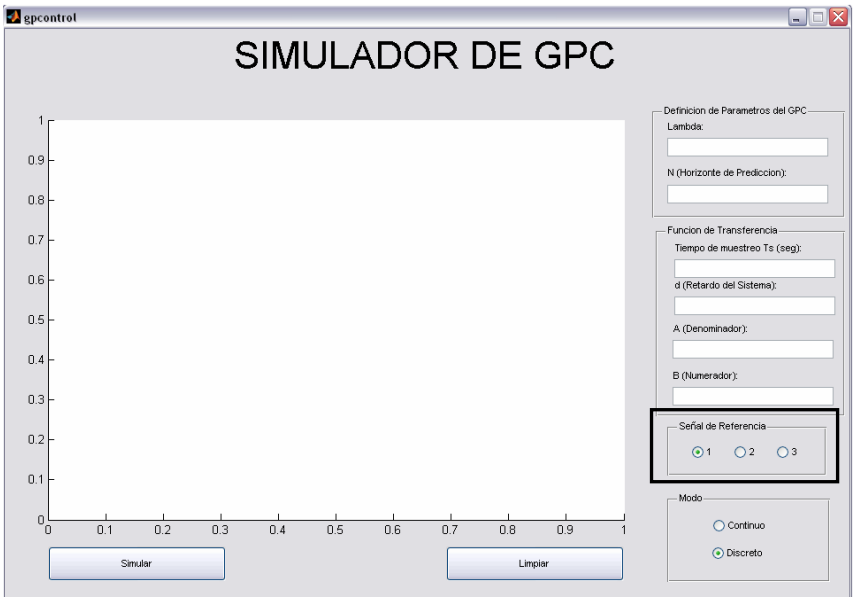

Figura 14. Simulador GPC (Trayectoria de referencia)

Una vez que se han introducido estos parámetros en cada una de las casillas mencionadas anteriormente, le damos clic en el botón simular y podremos observar como se comporta el sistema. Si queremos volver a utilizar el programa, le damos limpiar y automáticamente se borran todas las variables.

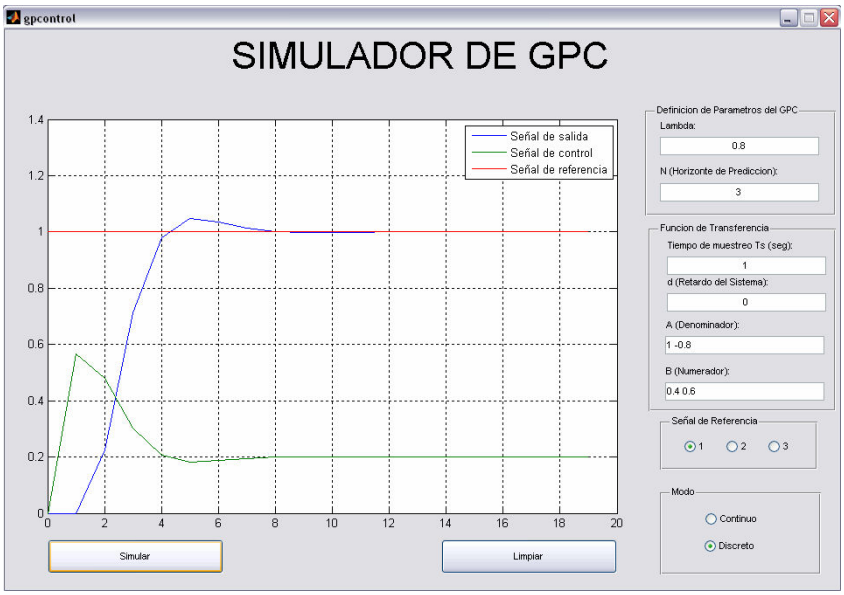

Figura 15. Simulador GPC

Los resultados de la simulación muestran el comportamiento del sistema en lazo cerrado. Como el objetivo de esta interfaz grafica es mirar que efectos sobre el modelo tiene variar λ, N y d. Vamos a realizar una serie de simulaciones con el GPC.

Empezaremos variando lambda, dejando fijos los demás parámetros, luego vamos a variar N y por ultimo variaremos d.

Para λ=0.5, tenemos la siguiente grafica:

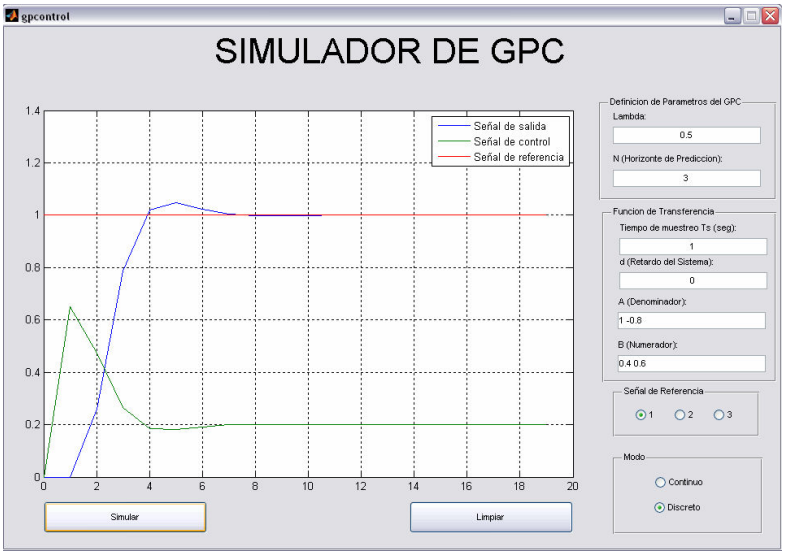

Figura 16. Ejemplo de aplicación en GUI cuando λ=0.5

Para λ=1, tenemos la siguiente grafica:

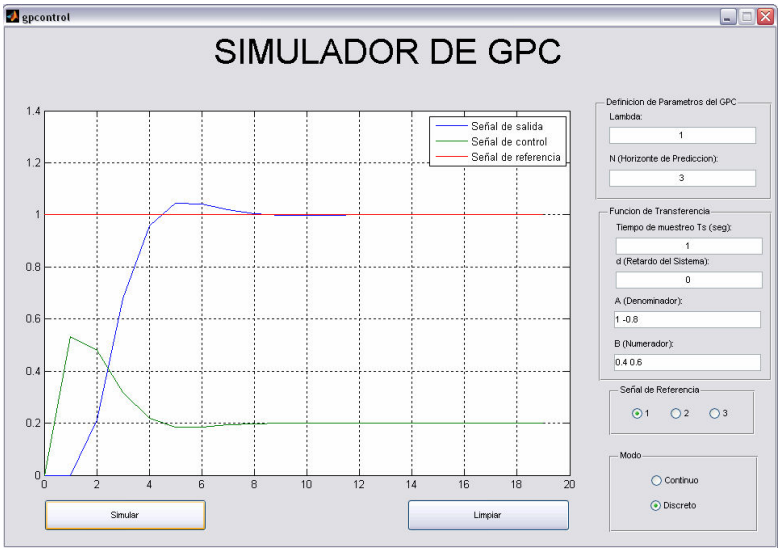

Figura 17. Ejemplo de aplicación en GUI cuando λ=1

Para λ=1.5, tenemos la siguiente grafica:

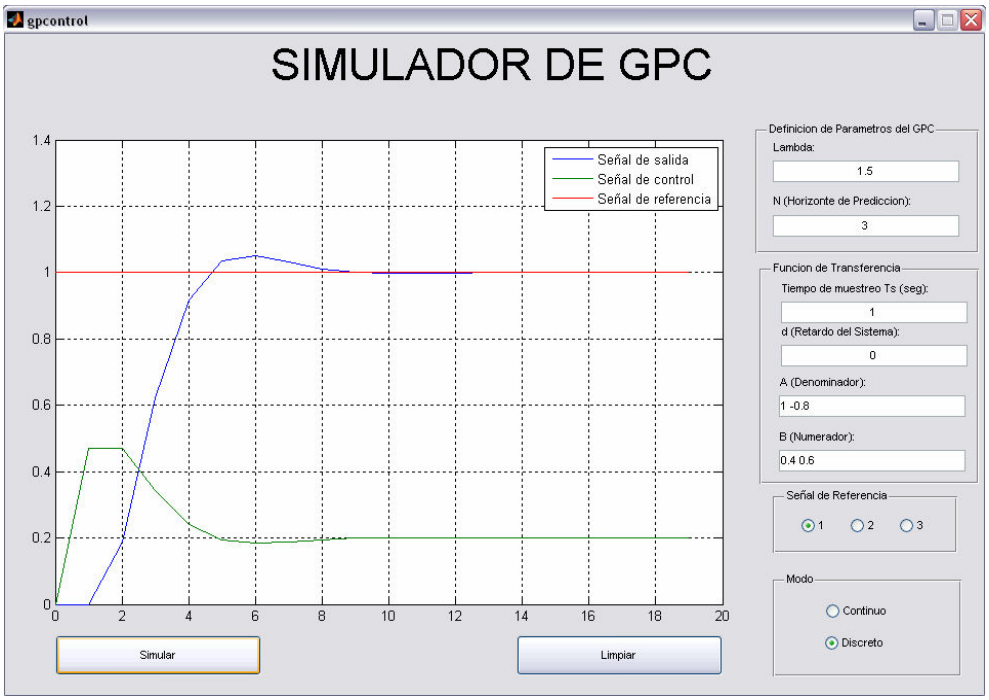

Figura 18. Ejemplo de aplicación en GUI cuando λ=1.5

Para λ=2, tenemos la siguiente grafica:

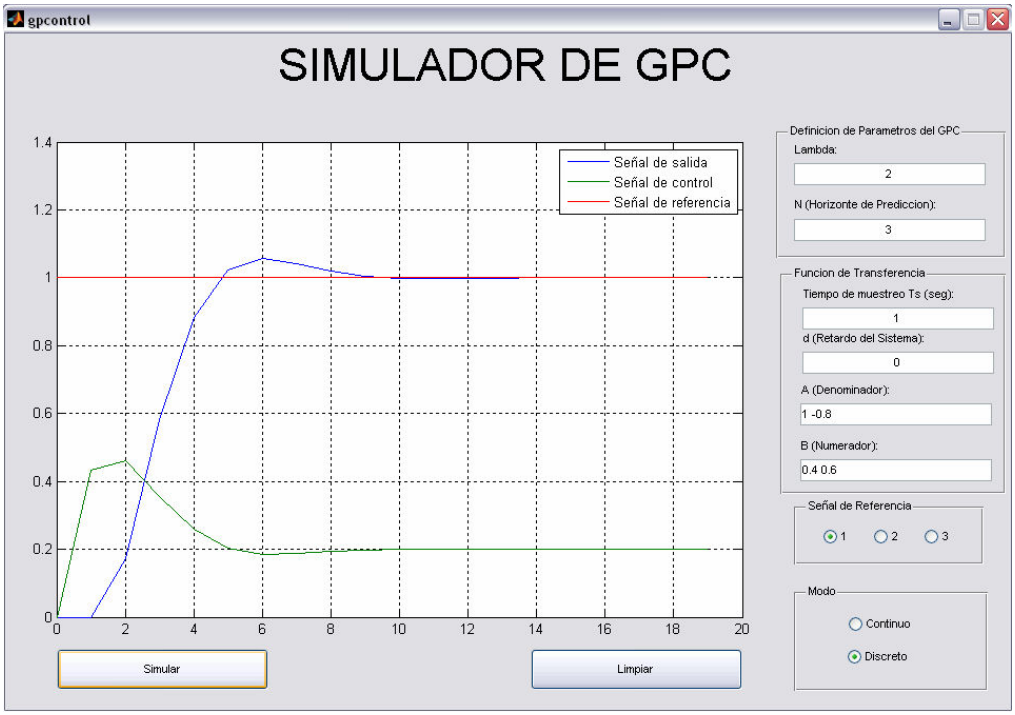

Figura 19. Ejemplo de aplicación en GUI cuando λ=2

Para λ=2.5, tenemos la siguiente grafica:

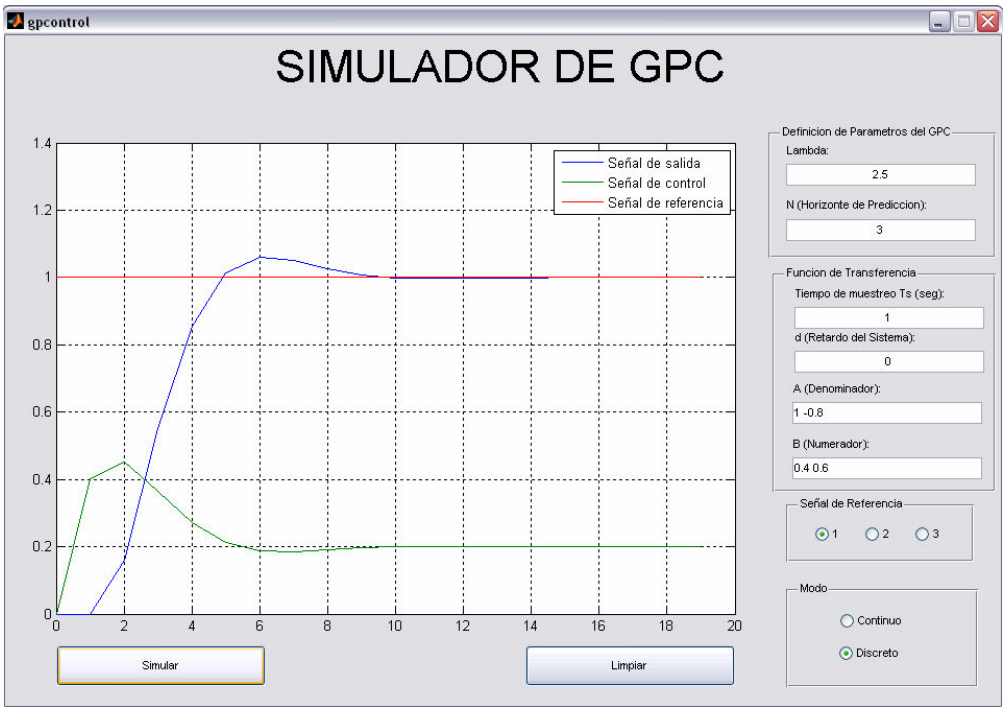

Figura 20. Ejemplo de aplicación en GUI cuando λ=2.5

Para λ=3, tenemos la siguiente grafica:

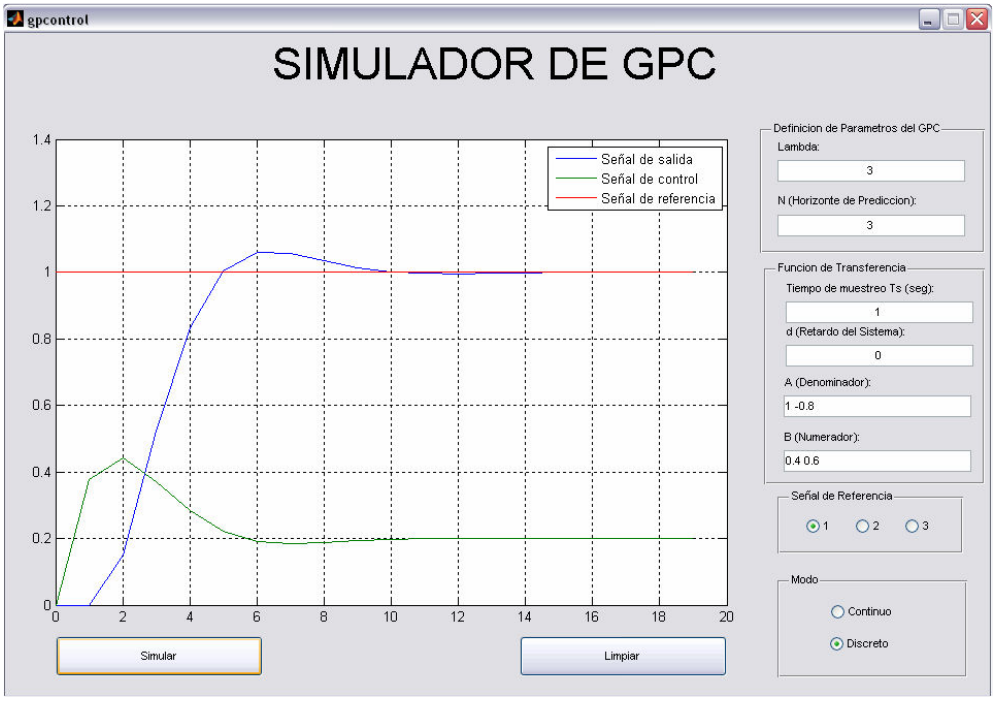

Figura 21. Ejemplo de aplicación en GUI cuando λ=3

Variaciones de las señales de control y de salida al cambiar el peso para la entrada en la función de costo lambda (λ)

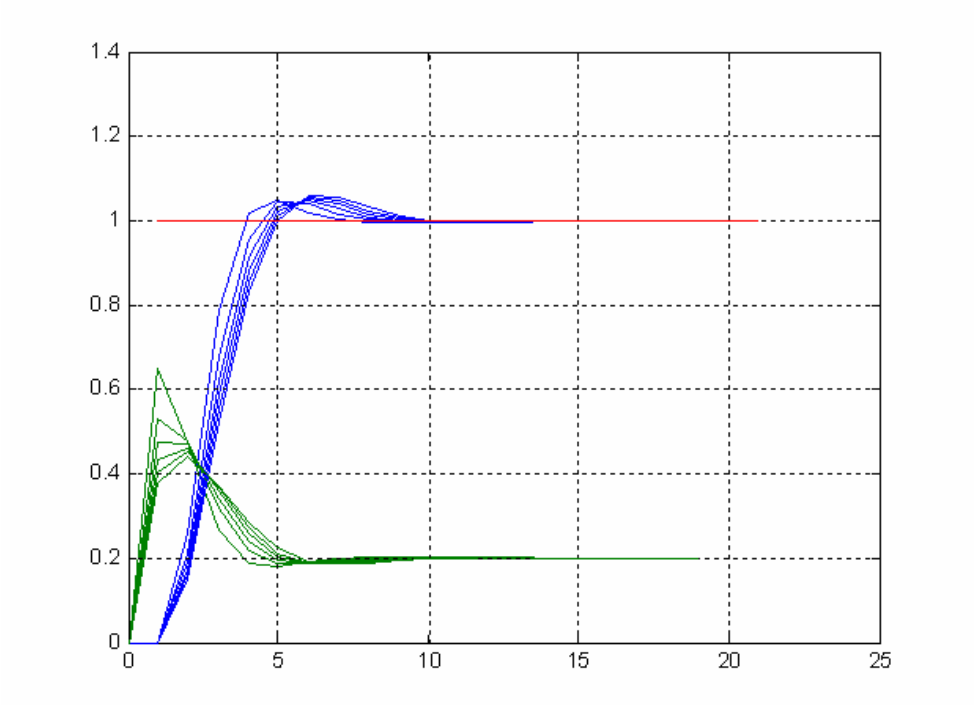

Figura 22. Ejemplo de aplicación en GUI variando lambda

Podemos concluir que al aumentar el valor de lambda, en la función de costo aumenta el coeficiente de ponderación de u, de manera que entre mayor sea lambda mayor es el nivel de minimización y la función de costo se ve incrementada, por lo que u (la función de control) se hace mas pequeña, como se ve en la grafica en la cual u parte de un valor pico aproximado de 0.6 y llega a un valor pico de 0.4 aproximadamente. Como la función de control (u) se hace mas pequeña entonces en la medida que se reduce este se retarda más la salida (y), lo cual se puede observar en la misma grafica.

A continuación vamos a variar N, dejando los demás parámetros fijos.

Para N=1, tenemos la siguiente grafica:

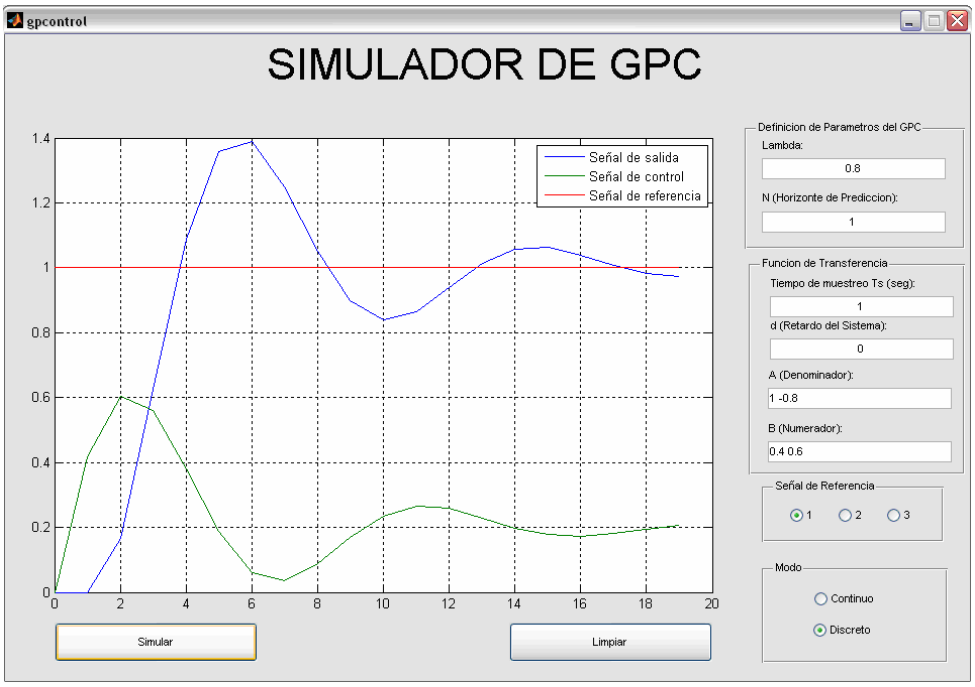

Figura 23. Ejemplo de aplicación en GUI cuando N=1

Para N=2, tenemos la siguiente grafica:

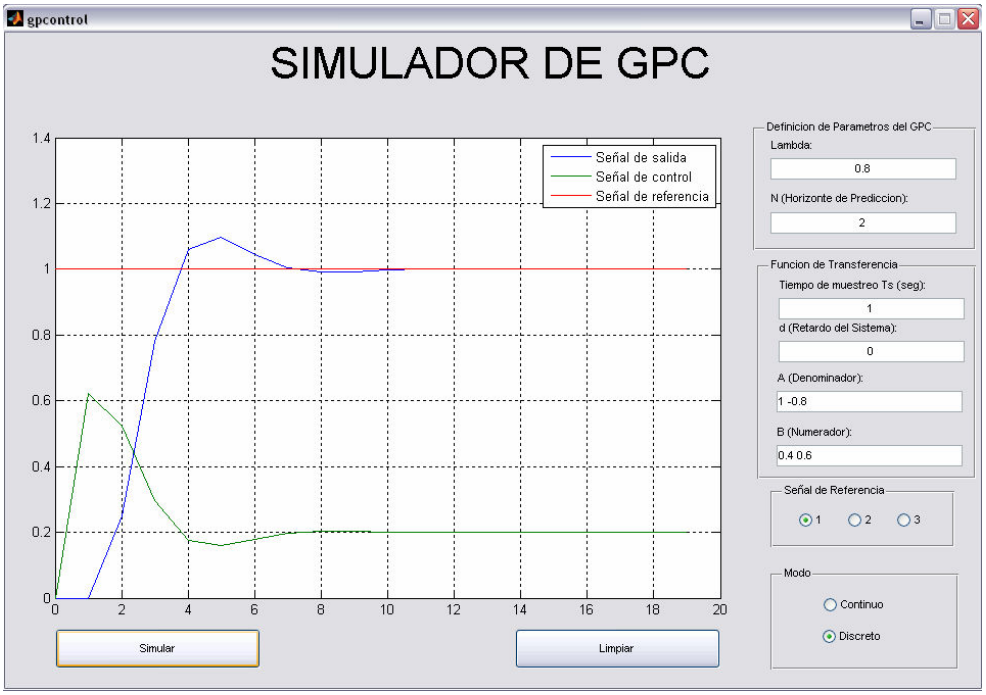

Figura 24. Ejemplo de aplicación en Matlab cuando N=2

Para N=3, tenemos la siguiente grafica:

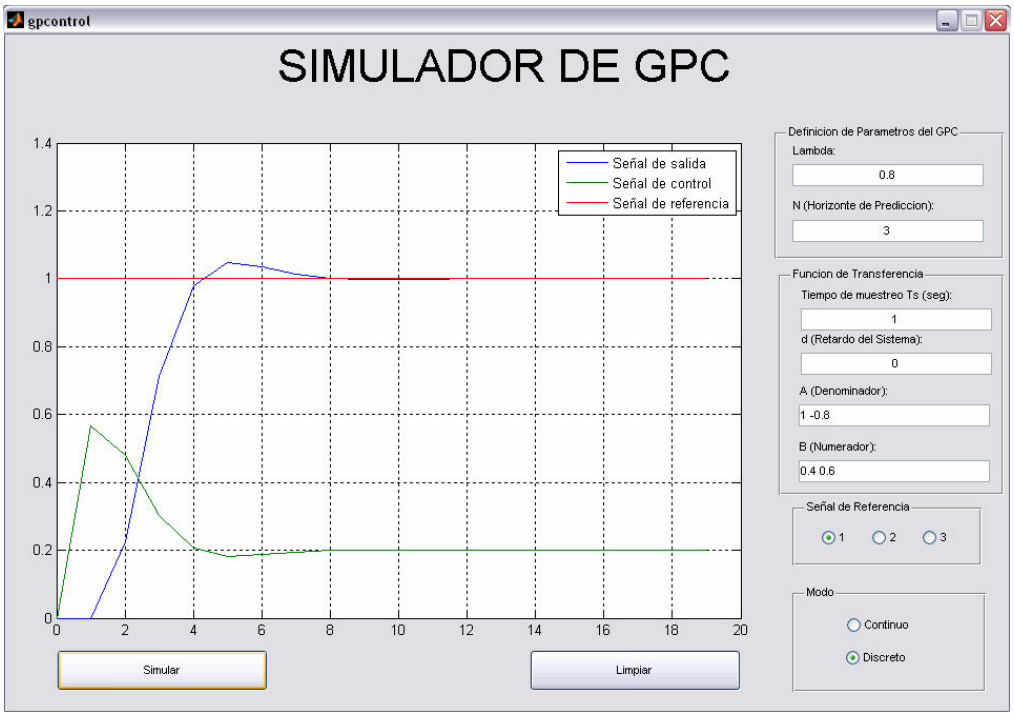

Figura 25. Ejemplo de aplicación en GUI cuando N=3

Para N=4, tenemos la siguiente grafica:

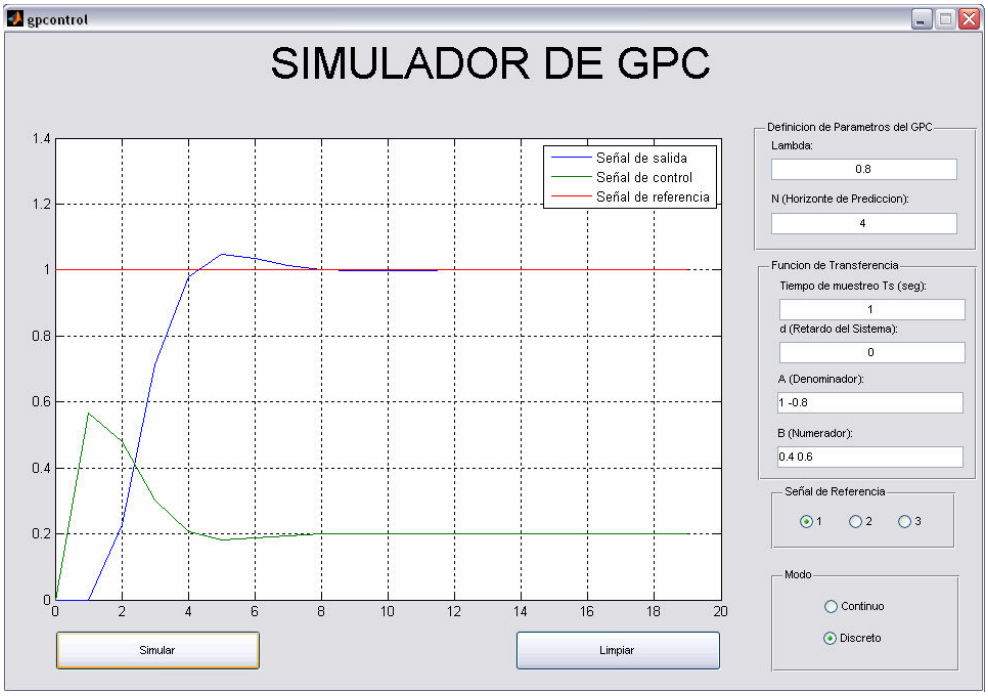

Figura 26. Ejemplo de aplicación en GUI cuando N=4

Para N=5, tenemos la siguiente grafica:

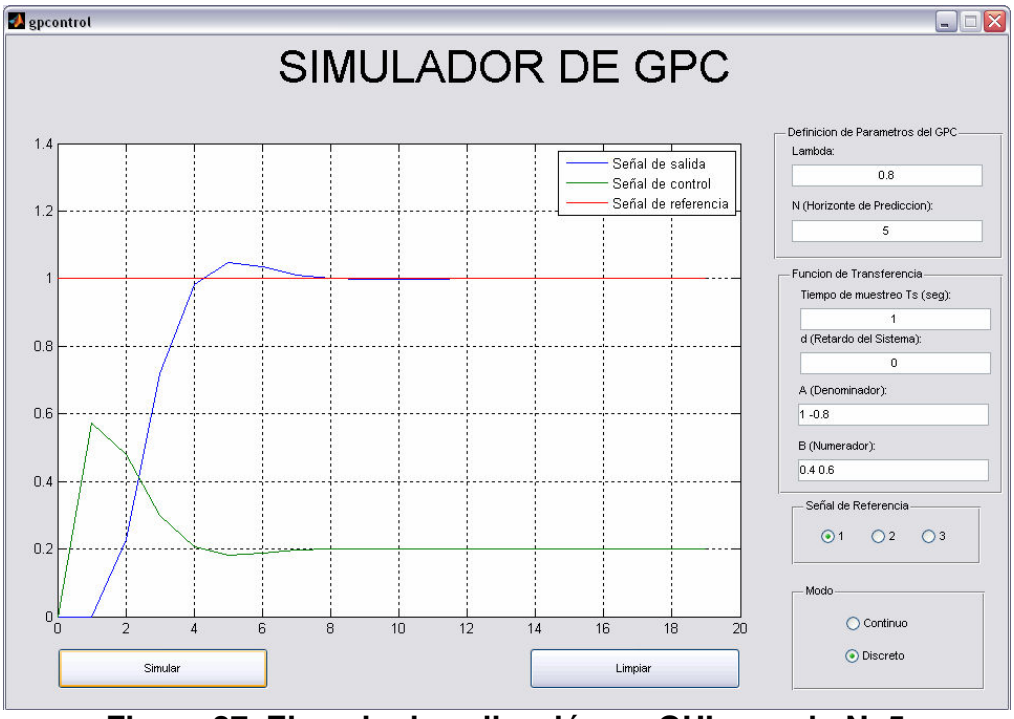

Figura 27. Ejemplo de aplicación en GUI cuando N=5

Variaciones de las funciones de control y de salida al cambiar el horizonte de predicción (N)

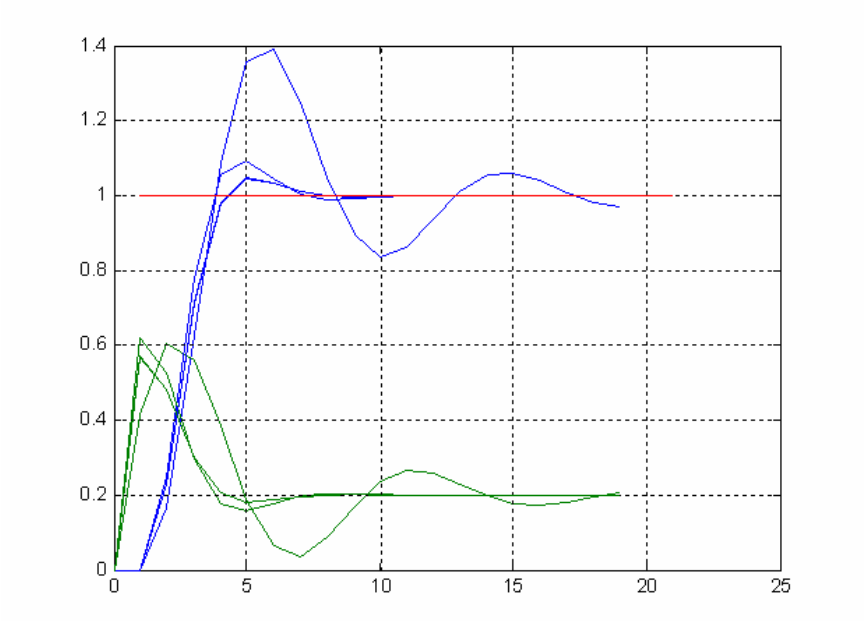

Figura 28. Ejemplo de aplicación en Matlab variando N

En la anterior grafica vemos que al aumentar el horizonte de predicción (N), aumenta el valor de la segunda sumatoria, al tener mas términos, y también aumenta el valor de la función de costo, que al hacerse mas grande recibe un mayor efecto en la minimización, si se tienen los casos extremos de valores de función de costo 0 y un valor muy grande, el valor 0 (el mínimo posible) no requeriría una minimización y para un valor muy grande la minimización se hace también considerablemente mas grande. Así la respuesta paso se hace mejor al incrementar N, sin embargo después de N = 3, los cambios son muy poco significativos.

A continuación vamos a proceder a variar d.

Para d=0 y N=3, tenemos la siguiente grafica:

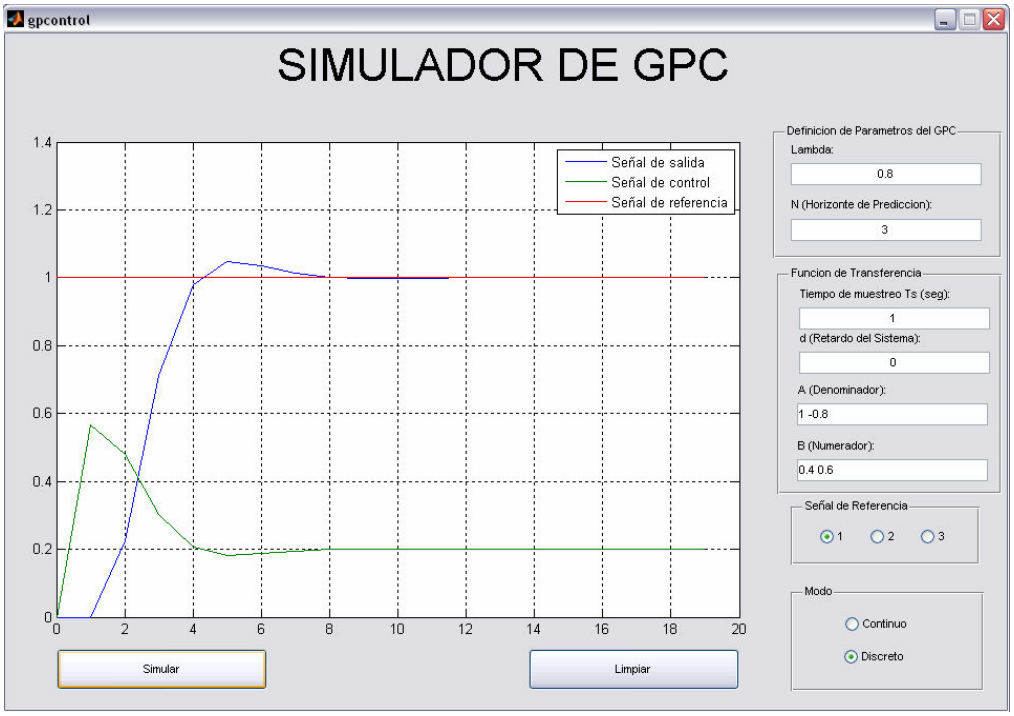

Figura 29. Ejemplo de aplicación en GUI cuando d=0 y N=3

Para d=1 y N=3, tenemos la siguiente grafica:

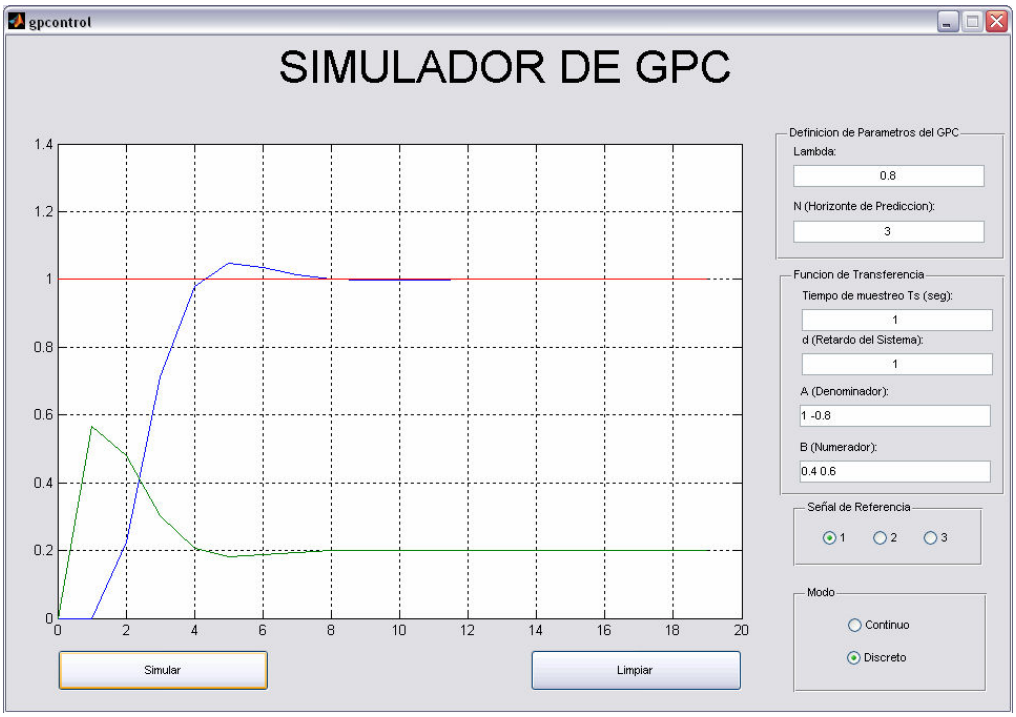

Figura 30. Ejemplo de aplicación en GUI cuando d=1 y N=3

Para d=2 y N=3, tenemos la siguiente grafica:

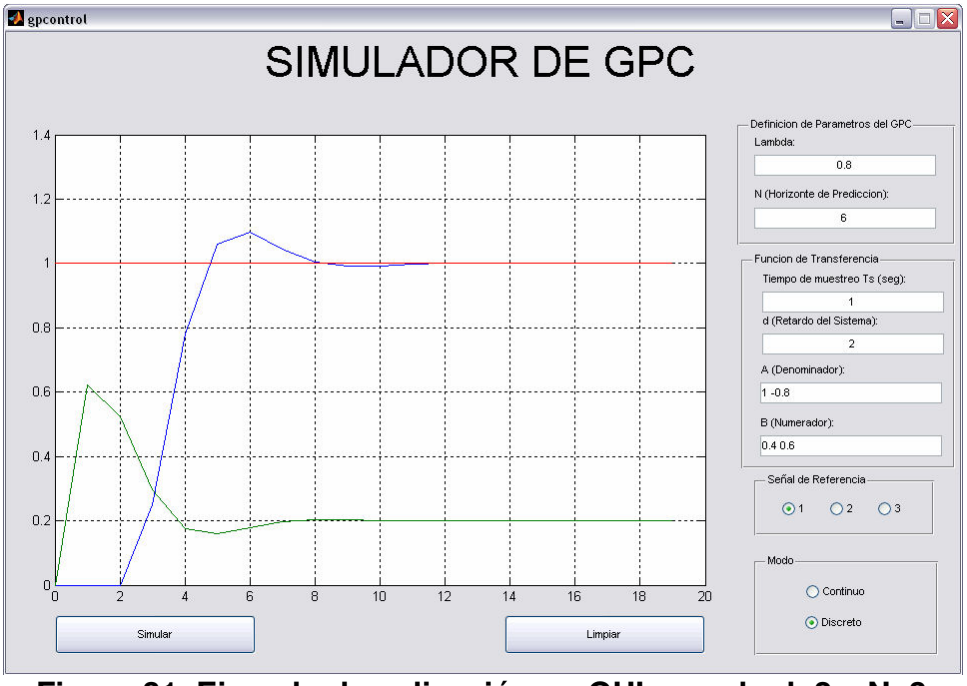

Figura 31. Ejemplo de aplicación en GUI cuando d=2 y N=3

Para d=3 y N=6, tenemos la siguiente grafica:

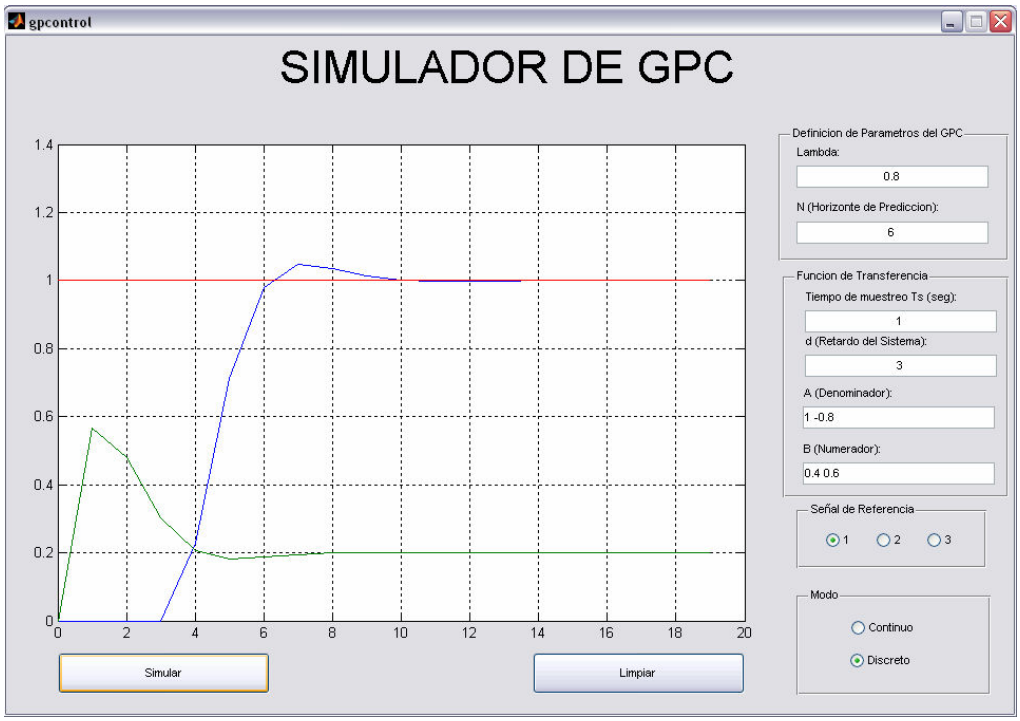

Figura 32. Ejemplo de aplicación en GUI cuando d=3 y N=6

Para d=4 y N=6, tenemos la siguiente grafica:

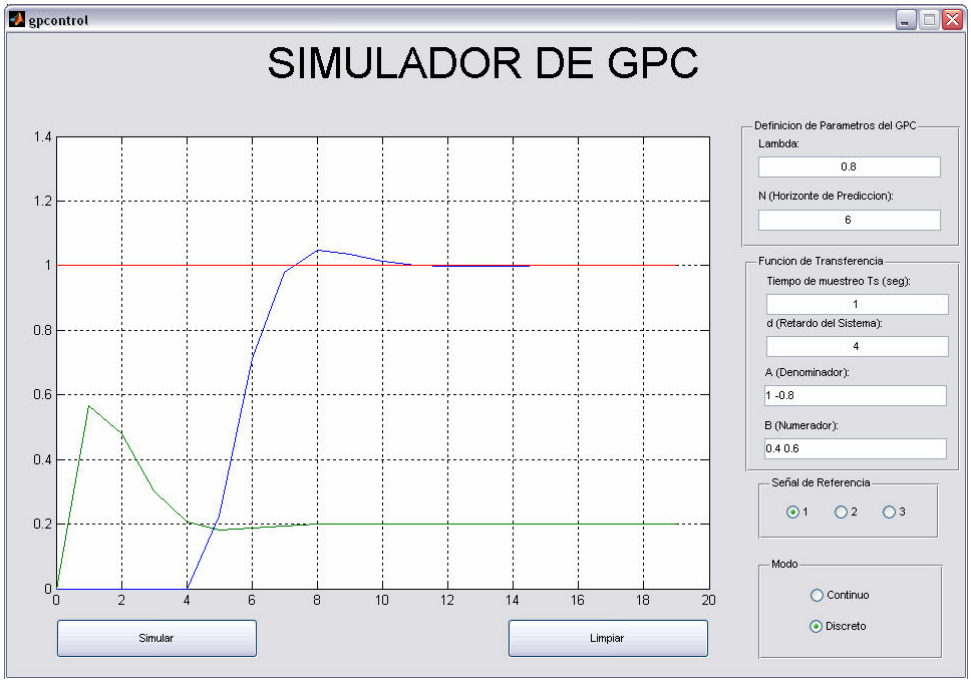

Figura 33. Ejemplo de aplicación en GUI cuando d=4 y N=6

Para d=5 y N=6, tenemos la siguiente grafica:

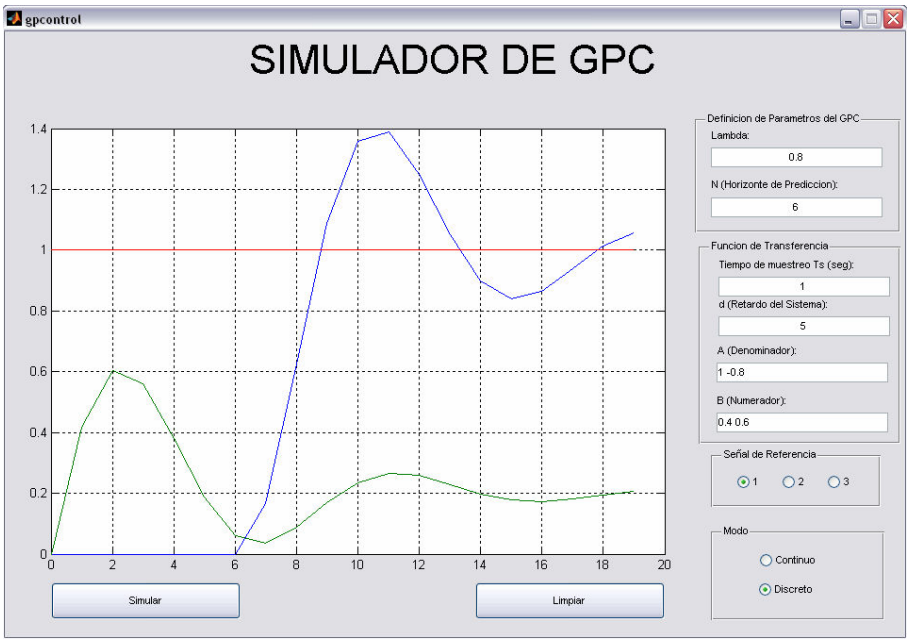

Figura 34. Ejemplo de aplicación en GUI cuando d=5 y N=6

Variaciones de las funciones de control y de salida al cambiar el retardo (d)

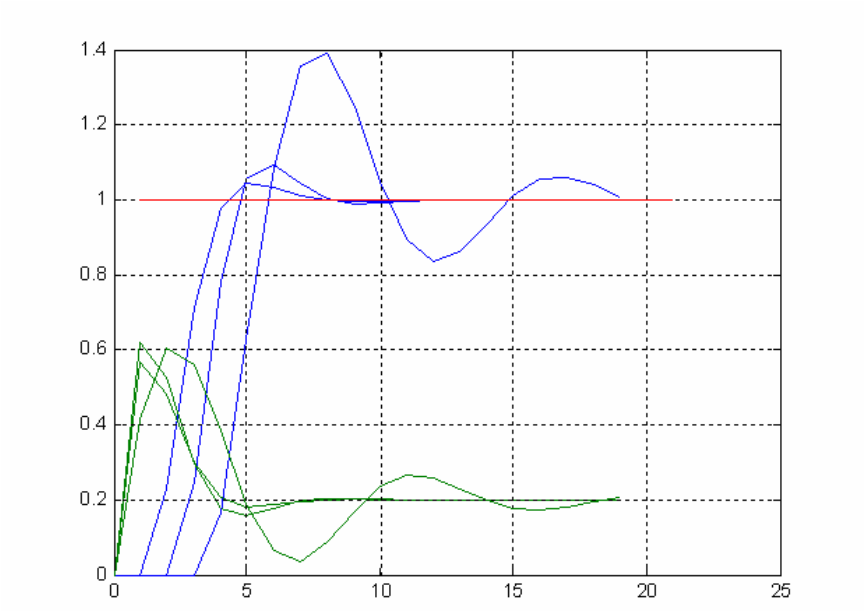

Figura 35. Ejemplo de aplicación en GUI variando d con N=3 y N=6

Al examinar las graficas correspondientes a los diferentes retardos, vemos que al disminuir la diferencia de tiempo entre un horizonte de predicción dado y los

diferentes valores de respuesta se empeora la función de salida. Como al tenerse un periodo de muestreo, la salida solo se vera afectada por la señal de control tras d+1 periodos, y la salida se mantiene constante después del horizonte de predicción, entonces al aumentar el retardo se reduce el tiempo de operación del controlador y se deteriora la calidad de la respuesta (señal o función de salida).

4. CONCLUSIONES

Se realizó el diseño de una interfaz grafica de usuario de forma sencilla y didáctica con el objetivo de que el usuario pueda variar los parámetros principales de los controladores predictivos y darse cuenta, la influencia que tienen estos cambios en la señal de salida.

Las herramientas que posee matlab para el desarrollo de GUIs, son muy poderosas y nos permiten elaborar programas que pueden interactuar con el usuario de una forma más amigable y ahorrarnos tiempo en la simulación de estos procesos.

Se han realizado pruebas con diferentes valores del horizonte de predicción pudiendo señalar que el punto objetivo se alcanza con cualquier valor de λ, pero la forma de alcanzarlo en cada caso es diferente. Dado que la salida del proceso al final del horizonte de predicción debe ser igual al punto objetivo, cuando el intervalo es corto implica una condición más exigente que requiere una acción de control mayor, la cual se reduce cuando el horizonte de predicción aumenta. A medida que aumentamos λ, la respuesta del sistema se hace más lenta.

Una de las ventajas de este programa es que permite visualizar los datos de una manera grafica lo que mejora la comprensión y el entendimiento de hacia donde se quiere llegar con un diseño especifico.

Con el desarrollo de éste programa, se pueden realizar nuevos proyectos para mejorar el diseño del software buscando implementar nuevos algoritmos. Se puede pensar en implementar la opción de escoger entre los diferentes algoritmos con los que trabaja el control predictivo y así el usuario poder escoger el algoritmo que mejor se adapte a sus necesidades.

El control predictivo constituye hoy en día una de las herramientas más poderosas para el control de procesos tecnológicos industriales. Su estructura interna, sustentada en un proceso de optimización permite predecir el desempeño futuro de la planta, así como su sistema de mando basados simplemente en el conocimiento pasado y futuro de las variables que lo caracterizan.

El control predictivo, a pesar del esfuerzo de cálculo que involucra, es una potente herramienta de control de procesos complejos de enorme futuro.

BIBLIOGRAFÍA

CAMACHO, Eduardo. Model Predictive Control. Primera edición. Sevilla: Springer-Verlag, 1998. 293 p.

ROSSITER, J. A. Model Based Predictive Control. Primera edición. Florida: CRC Press, 2005. 332 p.

IKONEN, Enso. Advanced Process Identification And Control. Primera edición. New York: Marcel Dekker, 2002. 316 p.

MORARI, Manfred. Model Predictive Control. Primera edición. Florida: CRC Press, 2002. 284 p.

MARCHAND, Patrick. Graphics And GUIs With Matlab. Tercera edición. New York: CRC Press, 2003. 521 p.

"MODEL PREDICTIVE CONTROL TOOLBOX USER'S GUIDE". The Math Works Inc, 2007.

"CONTROL SYSTEM TOOLBOX USER'S GUIDE". The Math Works Inc, 2007.

"MATLAB CREATING GRAPHICAL USER INTERFACES". The Math Works Inc, 2007.

"LEARNING MATLAB 7". The Math Works Inc, 2007.

Control Predictivo - Pasado, Presente y Futuro. Disponible en Internet en: http://riai.isa.upv.es/CGI-BIN/articulos%20revisados%202004/surveys/num3/MPC_Camacho_y_Bordons1.pdf

Control Predictivo: metodología, tecnología y nuevas perspectivas. Disponible en Internet en:
http://www.esi2.us.es/~bordons/apuntes_MPC.pdf

ANEXOS

ANEXO A. FUNCIONES DE MATLAB PARA LA CONSTRUCCION DE GUI's

De las muchas funciones que posee la herramienta computacional Matlab para el desarrollo de interfaces graficas de usuario, solo unas cuantas son usadas en la construcción de interfaces graficas, pero dado el beneficio obtenido mediante su uso es conveniente conocer la forma como operan y la forma como retornan sus valores de salida. A continuación se va a dar una breve descripción de la función que realiza cada una de ellas:

- **String** El texto mostrado en casi todos los controles (botones, botones de opción o selección, cajas de texto, listas de selección, menús pop-up)
- **Label** Propiedad de *uimenu* que especifica el texto que aparece en el menú.
- **Tag** Un nombre interno para el objeto. No lo ve el usuario, pero se utiliza mucho en programación para localizar un determinado objeto. GUIDE asigna un ***Tag*** por defecto a cada objeto que se crea. Este nombre puede respetarse o ser sustituido por otro elegido por el usuario.
- **Style** La clase de objeto de que se trata (*pushbutton* | *togglebutton* | *radiobutton* | *checkbox* | *edit* | *text* | *slider* | *frame* | *listbox* | *popupmenu*).
- **Position** Vector de cuatro elementos que indican la posición y tamaño del control [*left*, *bottom*, *width*, *height*]. Recuérdese que el origen está en la esquina inferior izquierda. Las unidades se expresan mediante la propiedad ***Units***. Extent Vector de cuatro elementos que indica el tamaño del *String* de un objeto o elemento (etiqueta o texto mostrado).
- **Units** Unidades en que se miden las dimensiones: *normalized* miden desde (0,0) a (1.0,1.0). También *pixels*, *inches*, *centimeters*, y *points*, que son unidades absolutas.
- **BackgroundColor** Vector de tres números entre 0 y 1.0 que indican las componentes RGB del color de fondo de un objeto.
- **ForegroundColor** Vector de tres números entre 0 y 1.0 que indican las componentes RGB del color del texto de un objeto.
- **Parent** El *handle* de la figura o control padre.
- **Children** Los *handle* de los controles hijos.
- **Enable** Si el control está activo o no, es decir si el usuario puede o no actuar sobre dicho control.
- **Visible** Si la figura o el control es visible o no.
- **FontName** El tipo de letra que se desea utilizar: 'Times New Roman', 'Arial', 'Courier New', etc.
- **FontSize** El tamaño de la letra en puntos (*points*).
- **FontWeight** Uno de estos valores: 'normal', 'ligth', 'demi' y bold'
- **UserData** Cualquier dato que el usuario quiera asociar con el control. No es utilizado por MATLAB. Los valores de *UserData* se pueden escribir con *set()* y leer con *get()*.

- **Value** Valor asociado con algunos controles: posición del cursor en la barra de desplazamiento, valor de la propiedad *max* cuando están en *on* y *min* cuando están en *off* en los *checkbuttons* y *radiobuttons*; número ordinal (empezando por 1) del elemento seleccionado en las *listbox* y *popupmenu*. Las cajas de texto, los botones y los frames no tienen esta propiedad.
- **gcf (*get current figure*)** devuelve el *handle* a la figura activa
- **gca (*get current axes*)** devuelve el *handle* de los ejes activos
- **gcbo (*get callback object*)** devuelve el *handle* del objeto sobre el que se ha producido el evento al que está tratando de responder el ***callback***
- **gcbf (*get callback figure*)** devuelve el *handle* de la figura sobre la que se ha producido el evento
- **findobj(gcbf, 'prop', 'propvalue')** devuelve el *handle* de un objeto de la figura activa cuya propiedad ***prop*** tiene el valor ***propvalue***. Los valores admitidos para la propiedad son los mismos que admite la función ***set***
- **get(handle, 'prop')** devuelve el valor de la propiedad *prop* en el objeto cuyo *handle* se pasa como primer argumento
- **set(handle, 'prop', 'propvalue')** Establece el valor *propvalue* en la propiedad *prop* del objeto cuyo *handle* se pasa como primer argumento

ANEXO B. FUNCION GPC

```
function [Duk,f]= GPC(A,B,G,N,d,u,y,r,k,lambda,Ts)


H=(G'*G+lambda*eye(N));
Hinv=inv(H);
Du=[u(1);diff(u)'];
f=f_matrix(A,B,N,Du,y,k,Ts); % Calculo de la respuesta libre


K=Hinv*G';

N1=d+1;        % Horizonte minimo de costo
Nu=d+N;        % Horizonte maximo de costo

Duk=K(1,:)*(r(k+N1:k+Nu)-f);    % Salida de control Du(k)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%%%%%%%%%Funcion para Calcular la respuesta no forzada
%%%%%%%%%%%%%%%%%%%%

function f=f_matrix(A,B,N,Du,y,k,Ts)

% A = polinomio A(z^-1)= [a0 a1*z^-1 ... an*z^-1]
% B = polinomio B(z^-1)= [b0 b1*z^-1 ... bm*z^-m]
% N = Horizonte de Prediccion
% Du = Vector columna de Delta u  Du=[Du(t-m) Du(t-(m-1)) ... Du(t)]'
% y = Vector columna con las salidas del sistema hasta el instante actual
% t, y=[y(t-n) y(t-(n-1)) ... y(t)]'
% f = Respuesta libre, f=[f(k+1) ... f(k+N)]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% f_(j)=z*[1-(1-z^-1)*A(z^-1)]*f_(j-1)+z^(j-1)*B(z^-1)*Du(t)
%%%%%%%%%%%

A_tilde=conv([1 -1],A);          % (1-z^-1)*A(z^-1)
La_tilde=length(A_tilde);
fa=[1 zeros(1,La_tilde-1)]-A_tilde; % [1-(1-z^-1)*A(z^-1)]
numa=1;dena=[0 1];                  % 1/z^-1
sysa=tf(numa,dena,Ts,'Variable','z^-1');
sysA=tf(fa,1,Ts,'Variable','z^-1');
sysfa=series(sysa,sysA);            %  z*[1-(1-z^-1)*A(z^-1)]
[numfa,denfa]=tfdata(sysfa);
Lfa=length(numfa{1});
if numfa{1}(end)==0
    numfa{1}(end)=[];
    Lfa=Lfa-1;
```

```
end

Lfb=length(B);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fk(1)=y(k);
j=0;
for i=1:N
    j=j+1;
    if j<Lfa
        if k+j-Lfa>0
            fj=[fk(j:-1:1);y(k-1:-1:k+j-Lfa)];
        else
            fj=[fk(j:-1:1);y(k-1:-1:1);zeros(Lfa-2,1)];
        end
    else
        fj=fk(j:-1:j-Lfa+1);
    end
    if j<Lfb
        if k+j-Lfb>0
            Duj=[zeros(j,1);Du(k-1:-1:k+j-Lfb)];
        else
            Duj=[zeros(j,1);Du(k-1:-1:1);zeros(Lfb+1-k-j,1)];
        end
    else
        Duj=[zeros(Lfb,1)];
    end
    fk(j+1,1)=numfa{1}*fj+B*Duj;
end
f=fk(2:j+1);
```

ANEXO C. CODIGO PRINCIPAL

```
function varargout = gpcontrol(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gpcontrol_OpeningFcn, ...
                   'gui_OutputFcn',  @gpcontrol_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 gpcontrol is made visible.
function gpcontrol_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 gpcontrol (see VARARGIN)

% Choose default command line output for gpcontrol
handles.output = hObject;
%text(0.9, 0.9, 'Lambda \lambda')
% Update handles structure
handles.r=ones(40,1);

guidata(hObject, handles);

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


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

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


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

d=str2double(get(handles.edit3,'String'));          % retardo del sistema
1-5
Ts=str2double(get(handles.edit7,'String'));         % Tiempo de Muestreo
(seg)
N=str2double(get(handles.edit2,'String'));          % Horizonte de
Prediccion
lambda=str2double(get(handles.edit1,'String'));     % Peso para la
entrada u en la funcion de costo

if isnan(d)
    d=0;
end

if isnan(Ts)
    Ts=1;
end
Ts = 1;

if isnan(N)
    N=3;
end

if isnan(lambda)
    lambda=0.5;
end

A=str2num(get(handles.edit4,'String'));  % A(z^-1)
B=str2num(get(handles.edit5,'String'));   % B(z^-1)

% Si el usuario no ingresa alguno de los polinomios de la fucion de
% transferencia el programa le muestra un mensaje de advertencia
if any([isempty(A) isempty(B)])
    warndlg('Ingrese ambos polinomios A y B','Not enough data');
end

x = get(handles.radiobutton1,'Value');
if x
    G = tf(B,A);
    [B,A] = tfdata(c2d(G,Ts),'v');
end
B = cat(2,zeros(1,d),B);
Gz=tf(B,A,Ts,'Variable','z^-1');  % G(z^-1)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[g,t]=step(Gz);
gk=g(1:N);
G=zeros(N);
for i=1:N
    G(i:N,i)=gk(1:N+1-i);
end

%%%%%  Reference trajectory  %%%%%%%%

r=handles.r;
yo=0;

y(1)=yo;
y(2)=0;
u(1)=0;
for k=2:20
    [Du,f]= GPC(A,B,G,N,d,u,y,r,k,lambda,Ts);
    u(k)=u(k-1)+Du;
    [yk,t]=lsim(Gz,u,[],y(1));
    y(k+1)=yk(end);
end

axes(handles.axes1);
plot(t,y(1:20)',t,u');
hold on
plot(r(1:21),'r')
grid
hold off

legend('Señal de salida','Señal de control','Señal de referencia')
%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%



% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
cla(handles.axes1,'reset');



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

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

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


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

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

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

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

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

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


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

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

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

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

function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (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 edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a
double

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

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

function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (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 edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a
double

% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

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

function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (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 edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7 as a
%        double

% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

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

% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton1

% --- Executes when selected object is changed in uipanel2.
function uipanel2_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in uipanel2
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
switch get(hObject,'Tag')   % Get Tag of selected object
    case 'radiobutton3'
        handles.r = ones(40,1);
    case 'radiobutton4'
        handles.r = sin(0.25*(1:40))';
    case 'radiobutton5'
        handles.r =sawtooth(0.66*(0:40))';
end

guidata(hObject, handles);
```