Graphische Oberflächen, erzeugt mit Matlab

1. Controls in der graphischen Matlab-Oberfläche

1.1 Liste der Controls

Controls umfassen folgende Objekte:

`pushbutton`	um eine Aktion durch einen Mausclick auszulösen.
`checkbox`	als einzelnes Objekt oder in einer Gruppe (auf einem Frame) um mehrere Optionen zu setzen.
`radiobutton`	als einzelnes Objekt oder in einer Gruppe, um 1 Option zu setzen.
`slider`	um einen Wert aus einem Bereich auszuwählen
`popupmenu`	zur Auswahl aus einer Liste
`text`	um statischen Text am Bildschirm anzuzeigen.
`edit`	editierbares Textfeld.
`frame`	rechteckiges Feld, um ein oder mehrere Controls logisch zu gruppieren.

1.2 Erzeugung von Controls im M-File

Der allgemeine Aufruf zur Erzeugung eines Controls ist wie folgt:

```
h = uicontrol(hfig, 'PropertyName',PropertyValue, ...)
```

Das Control wird durch Hintereinanderreihen von verschiedenen `'PropertyName', PropertyValue` beschrieben. Der Parameter `hfig` bezeichnet den Handle der Figur (Fenster), in dem das Control platziert werden soll. `h` ist der Handel des Controls.

Die Aktion, die ausgelöst werden soll, wenn der Benutzer auf das Control clickt, wird in der Control-Eigenschaft `CallBack` (das ist eine Matrix mit den auszuführenden Befehlen als Strings) beschrieben. Für ein editierbares Textfeld (`text`) wird dessen `CallBack` von Matlab aufgerufen, wenn der Benutzer

- das Textfeld mit der Maus verlässt
- bei einzeiligem Text Return drückt
- bei mehrzeiligem Text Cmd-Return oder Enter drückt.

Beispiel:

Ein pushbutton zeigt den Wert eines anderen Controls, nämlich eines sliders an. Der Handle des sliders ist in `sli8` festgelegt. Die Position des pushbutton `buttongetval` ist im aktuellen Graphikfenster (`gcf`) von links unten gerechnet an der Stelle [10 120 120 25] (in Pixel).

```
buttongetval = uicontrol(gcf,...
            'Style', 'push', ...
            'Position', [10 120 120 25],...
            'String', 'Get Slider Value',...
            'CallBack', 'disp(get(sli8,''value''))');
```

1.2 Abfragen von Controls

Eigenschaften von Controls (Handle des angesprochenen Controls `hcontrol`) können mit

```
PropertyValue = get(hcontrol, 'PropertyName');
```

abgeftragt werden.

1.3 Eigenschaften von Controls

Die momentanen Eigenschaften eines Controls können mit `get(hcontrol)` aufgelistet werden. `hcontrol` ist der Handle des Controls, der bei dessen Erzeugung returniert wird.

Der Befehl `set(hcontrol)` listet alle Control-Eigenschaftsnamen und alle dazu möglichen Eigenschaftswerte auf.

Die Eigenschaft `'Style'`:

Mögliche Werte für diese Eigenschaft sind `'checkbox'`, `'edit'`, `'frame'`, `'popupmenu'`, `'pushbutton'`, `'radiobutton'`, `'slider'`, `'text'`.

Die Eigenschaft `'String'`:

- für die Objekte checkbox, pushbutton, radiobutton wird hier das Label angegeben, das auf dem Control erscheinen soll.
- für text wird hier der darzustellende statische Text angegeben.
- für edit wird hier der erstmals angezeigte und später vom Benutzer editierte String gespeichert.
- für popupmenus wird hier die Liste der Auswahlen gesetzt.

Die Eigenschaft `'HorizontalAlignment'`:

geht auf dem Macintosh nur für statischen Text und platziert diesen auf der Fläche (siehe `'Position'`) des Controls `'left'`, `'center'` oder `'right'`.

Die Eigenschaft `'Position'`:

wird von der linken unteren Bildecke des Fensters gerechnet:

```
'Position' [left, bottom, width, height]
```

Die Position wird entweder in Absolutwerten oder in Relativwerten angegeben (siehe Eigenschaft `'Unit'`).

Die Eigenschaft `'Unit'`:

Sie spezifiziert, in welcher Einheit die `'Position'`-Angabe zu verstehen ist:

`'pixels'` (Default-Einstellung), `'inches'`, `'centimeters'`, `'points'` (1 inch = 72 points) oder `'normalized'`. Bei `'normalized'` hat die linke untere Ecke des Fensters (figure) die Werte (0.0, 0.0) und die Ecke rechts oben (1.0, 1.0).

Falls es dem Benutzer erlaubt ist, das Fenster in der Grösse zu verändern (Eigenschaft des Fensters `'Resize', 'on'`), verschieben sich die Controls auf dem Fenster beim Resize nur mit, wenn sie mit `'Unit', 'normalized'` platziert wurden.

Die Eigenschaft `'CallBack'`:

Sie legt die Befehle fest, die Matlab ausführen soll, wenn das Control aktiviert wird. Aufruf: `'CallBack', 'string'`

Der `string` kann ein beliebiger oder mehrere Matlab-Befehle, ein M-file Name oder ein Variablenname sein.

Wenn das Control aktiviert wird (z.b. click auf einen pushbutton), übergibt Matlab den `string` an die Funktion `eval` und führt ihn aus, als ob er als Befehl vom Command-Fenster eingegeben würde. Der `CallBack` wird in dem Matlab Workspace ausgeführt. Somit stehen dem `CallBack` alle Variablen des Workspace zur Verfügung.

Die Eigenschaft `'Value'`:

beinhaltet den aktuellen Wert eines Controls. Er wird normalerweise von Matlab gesetzt, kann aber auch per Befehl gesetzt werden.

```
z.B. set(hcontrol, 'Value', value);
```

Je nach Control beinhaltet `value` verschiedene Daten:

- pushbutton: Ist der pushbutton gedrückt, steht in `value` der Wert der Eigenschaft `'Max'`. Ist der Callback vollständig ausgeführt, beinhaltet `value` den Wert von `'Min'`.
- checkbox, radiobutton: `value` hat den Wert von der Eigenschaft `'Max'`, falls das Control selektiert (`'on'`) ist, sonst (`'off'`) hat `value` den Wert von `'Min'`.
- slider: `value` beinhaltet den Wert der Stelle, an der Schieber steht.
- popupmenu: `value` ist der Index des ausgewählte Eintrags aus der Liste, von oben nach unten gezählt (Benutzer wählt den 2. Eintrag => `value = 2`).

Die Eigenschaften `'Min'` und `'Max'`:

Sie beschreiben die zulässigen Wertegrenzen für das Control.

```
'Max', 'maxvalue'
'Min', 'minvalue'
```

Die Default-Werte sind `minvalue = 0`, `maxvalue = 1`.

Diese Eigenschaften haben folgende Bedeutung:

- pushbutton: `maxvalue` wird angenommen, falls der pushbutton gedrückt ist (das ist also während der Ausführung des Callback), `minvalue` sonst.
- checkbox, radiobutton: `maxvalue`, wenn selektiert, sonst `minvalue`.
- slider: `minvalue` und `maxvalue` beschreiben den auswählbaren Bereich des sliders.
- edit: ein editierbare Text ist nur einzeilig, falls `maxvalue - minvalue` ≤ `1`. Er ist mehrzeilig, falls `maxvalue - minvalue > 1`. Die Default-Einstellung ist einzeilig. Wird der Text als mehrzeilig definiert, kann der Benutzer soviele Zeilen eingeben wie ihm beliebt (`maxvalue` beinhaltet NICHT die maximale Anzahl Zeilen).

Die Eigenschaft `'UserData'`:

```
'UserData', matrix
```

Durch die Eigenschaft `'UserData'` können in der `matrix` beliebige Daten dem Control zur Verfügung gestellt werden.

Werden z.B. die Handles von anderen Controls in `matrix` gespeichert, so kann dieses Objekt beim `CallBack` auf die anderen Controls zugreifen und deren Eigenschaften abändern.

Die Eigenschaft `'Visible'`:

`'Visible', 'on'` bestimmt, dass ein Control (oder auch ein Fenster (`figure`)) sichtbar ist. `'Visible', 'off'` versteckt das Control bzw. Fenster. Diese Eigenschaft kann beispielsweise genutzt werden, um abhängig von der Benutzerauswahl in einem Fenster zusätzliche Controls anzubieten.

1.4 Programmieren von radiobuttons

radiobuttons sind gedacht, um dem Benutzer die Auswahl von genau einer Option (also eines radiobutton) aus einem Set von sich gegenseitig ausschliessenden Optionen zu treffen. Hier in der Matlab-Umgebung ist der Programmierer dafür verantwortlich, dass alle anderen radiobuttons in einem Set zurückgesetzt werden, wenn ein radiobutton vom Benutzer ausgewählt wird.

Das Vorgehen dazu ist folgendermassen:

Die radiobuttons, die logisch zusammengehören und sich somit gegenseitig ausschliessen sollen, werden gruppiert (vgl. Beispiel unten: `sym(1)` ... `sym(3)`). In der `CallBack`-Anweisung wird der gegenseitige Ausschluss programmiert.

Beispiel:

Ein Benutzer soll aus einem radiobutton Set auswählen können, mit welchem Charakter ein plot vorgenommen werden soll: `"°"`, `"+"` oder `"*"`. Grundeinstellung sei `"°"`, also `'Value'` von `sym(1)` ist 1.

```
sym(1) = uicontrol(fig, 'Style', 'radio',...
            'Position', [100 300 75 25],...
            'String', 'Circle',...
            'Value', 1);
sym(2) = uicontrol(fig, 'Style', 'radio',...
            'Position', [100 275 75 25],...
            'String', 'Plus',...
            'Value', 0);
```

```
sym(3) = uicontrol(fig, 'Style', 'radio',...
           'Position', [100 250 75 25],...
           'String', 'Star',...
           'Value', 0);
```

Damit sich die drei radiobuttons gegenseitig ausschliessen, speichert jeder radiobutton in seinem Feld `'UserData'` die Handles der anderen beiden. Selektiert der Benutzer einen radiobutton, so wird dessen `'CallBack'` aufgerufen. Somit muss man also diesem Callback zufügen, dass er die anderen beiden radiobuttons prüft und gegebenenfalls auf 0 setzt. Der Zugriff auf die beiden anderen radiobuttons erfolgt über ihre Handles, die ja in den `'UserData'` gemerkt wurden.

```
% merken der Handels der anderen beiden radiobuttons
for i = 1:3
 set(sym(i), 'UserData', sym(:, [1:(i-1), (i+1):3]))
end;
% CallBack-Befehle in call = ...
call = [
 'me = get(gcf, ''CurrentObject'');', ...
 'if (get(me, ''Value'') == 1),', ...
      'set (get(me, ''UserData''), ''Value'', 0),', ...
 'else,', ...
      'set(me, ''Value'', 1),', ...
 end' ];
% setzen der Eigenschaft 'CallBack'
set(sym, 'CallBack', call);
```

2. Programmierhinweise

2.1 Gerüst eines UserInterface-Programms

Am besten packt man das gesamte UserInterface-Programm in eine M-Function. Sie erhält als Übergabeparameter die auszuführende Aktion als String. Innerhalb der M-Function wird dann in einer if-elseif-end Anweisung eine Fallunterscheidung für alle Aktionen vorgenommen. In dem Fall `'initialize'` werden z.B. alle Controls definiert. Ausserdem werden hier die nötigen Aktionen, die von den Controls ausgehen sollen in deren Callback festgelegt. Der Callback-String ist ein erneuter Aufruf dieser M-Function, aber mit der für den Control spezifischen Aktion.

```
function myfirstUI (action);

if nargin == 0
 action = 'initialize';
end

if strcmp(action, 'initialize')
      % create controls for the user interface
      h1 = uicontrol(gcf, ....
           'Style', 'pushbutton', ...
           'Position', [10 120 120 25],...
           'String', 'do first action',...
           'CallBack', 'myfirstUI(''firstaction'')');
```

```
        % similar h2 ...
        set(h2, 'CallBack', 'myfirstUI(''secondaction'')');

    elseif strcmp(action, 'firstaction'),
        % now do the actions associated with button h1
        disp('pushbutton h1 selected.')

    elseif strcmp(action, 'secondaction'),
        % now do the actions associated with button h2
        disp('pushbutton h2 selected.')

    end;
```

Meistens sind die Aktionen, die durch die Controls ausgelöst werden, nicht so einfach, wie oben angedeutet. Häufig möchte man als Aktion auf andere Controls zugreifen und deren Eigenschaften verändern (z. B. ein slider gibt seinen aktuell veränderten Wert einem edit oder text Control zur numerischen Anzeige weiter). Um solche Aktionen ausführen zu können, müssen die Handles der Controls natürlich in jedem erneuten Aufruf der Funktion `myfirstUI` bekannt sein. Das erreicht man entweder über globale Variablen oder durch Ausnutzung der Objekteigenschaft `'UserData'`. Im folgenden wird dies näher beschrieben:

2.2 Globale Variablen

Dies ist ein schneller und effizienter Weg, birgt aber mehrere Gefahren:

- Führt der Benutzer den Befehl `clear global` aus, werden alle Handles gelöscht.
- Falls dem Benutzer erlaubt wird mehrere Figurfenster zu öffnen, ist Matlab nicht mehr in der Lage die globalen Handles automatisch zu verwalten. Für jedes geöffnete Fenster müsste also ein separater Satz von globalen Variablen angelegt werden (wofür der Programmierer verantwortlich ist).

Folgendes Beispiel zeigt, wie die Handles auf die Controls in der globalen Variablen `UI_PB` und `UI_RB` gespeichert werden. Matlab empfiehlt GROSSBUCHSTABEN zur Kennzeichnung von globalen Variablen.

```
function mysecondUI (action);

global  UI_PB  UI_RB

if nargin == 0
 action = 'initialize';
end

if strcmp(action, 'initialize')
 % create controls for the user interface
 UI_PB = uicontrol(gcf, ....
      'Style', 'pushbutton', ...
      'Position', [10 100 120 25],...
      'String', 'do with your rb selection',...
      'CallBack', 'mysecondUI(''do'')');
 UI_RB(1) = uicontrol(gcf, ....
      'Style', 'radiobutton', ...
```

```
      'Position', [10 120 120 25],...
      'String', 'select this',...
      'CallBack', 'mysecondUI(''resetrb2'')');
 UI_RB(2) = uicontrol(gcf, ....
      'Style', 'radiobutton', ...
      'Position', [10 140 120 25],...
      'String', 'select that',...
      'CallBack', 'mysecondUI(''resetrb1'')');

elseif strcmp(action, 'resetrb1'),
 % set radiobutton UI_RB(1) to 0 and UI_RB(2) to 1

elseif strcmp(action, 'resetrb2'),
 % set radiobutton UI_RB(2) to 0 and UI_RB(1) to 1

elseif strcmp(action, 'do'),
 % now do the actions associated with button UI_PB
 if get(UI_RB(1), 'Value') == 1,
      % call this function
 elseif get(UI_RB(2), 'Value') == 1,
      % call that function
 end

% elseif strcmp...  (other callbacks)

end;
```

2.3 Ausnutzen der Matrix UserData (Objektattribut)

Sich die Objekt-Handles in einer `UserData` Matrix der Figur zu merken, ist zwar etwas komplizierter, hat aber folgende Vorteile:

- Die Handles existieren sicher solange wie das Figurfenster offen ist.
- Auch mehrere Fenster können verwaltet werden.

Das folgende Beispiel entspricht dem obigen. Die Handles des pushbuttons und der radiobuttons werden am Ende des `'initialize'` der Figur (`gcf`) in der Eigenschaft `'UserData'` als Matrix zugeordnet:

```
set(gcf, 'UserData', [pb rb])
```

Wird die Aktion `'do'` ausgeführt, müssen die Objekt-Handles erst wieder aus den `'UserData'` der Figur extrahiert werden.

```
ui_handles = get(gcf, 'UserData');
rb(1) = ui_handles(2);
rb(2) = ui_handles(3);
```

Diese Extraktion der Daten aus der Figur ist für jeden `'CallBack'` (also für jede Aktion ausser `'initialize'`) durchzuführen, der auf diese Daten zugreifen möchte.

Die gesamte Funktion sieht dann so aus:

```
function mysecondUI (action);
```

```
if nargin == 0
 action = 'initialize';
end

if strcmp(action, 'initialize')
 % create controls for the user interface
 pb = uicontrol(gcf, ....
      'Style', 'pushbutton', ...
      'Position', [10 100 120 25],...
      'String', 'do with your rb selection',...
      'CallBack', 'mysecondUI(''do'')');
 rb(1) = uicontrol(gcf, ....
      'Style', 'radiobutton', ...
      'Position', [10 120 120 25],...
      'String', 'select this',...
      'CallBack', 'mysecondUI(''resetrb2'')');
 rb(2) = uicontrol(gcf, ....
      'Style', 'radiobutton', ...
      'Position', [10 140 120 25],...
      'String', 'select that',...
      'CallBack', 'mysecondUI(''resetrb1'')');

 % store object handles
      set(gcf, 'UserData', [pb rb])

elseif strcmp(action, 'resetrb1'),
 % set radiobutton rb(1) to 0 and rb(2) to 1

elseif strcmp(action, 'resetrb2'),
 % set radiobutton rb(2) to 0 and rb(1) to 1

elseif strcmp(action, 'do'),
 % retrieve object handles from figur's 'UserData'
      ui_handles = get(gcf, 'UserData');
      rb(1) = ui_handles(2);
      rb(2) = ui_handles(3);

 % now do the actions associated with pushbutton pb
      if get(rb(1), 'Value') == 1,
           % call this function
      elseif get(rb(2), 'Value') == 1,
           % call that function
      end

% elseif strcmp...  (other callbacks)
end;
```

3. Arbeiten mit der Mausposition

Unter diesem Abschnitt wird erklärt, wie die aktuelle Mausposition abgefragt und auf Mausclicks reagiert werden kann. Für folgende Benutzeraktionen kann ein Callback programmiert werden:

- Drücken der Maustaste während sich die Maus innerhalb eines Figurfensters befindet;
- Loslassen der Maustaste;
- Bewegen der Maus innerhalb eines Figurfensters.

Ein Callback ist eine Anzahl von Befehlen, die von Matlab ausgeführt werden, sobald die entsprechende Benutzeraktion ausgelöst wird (zur Erinnerung: derjenige Callback eines pushbutton, bezeichnet als Eigenschaft `'CallBack'`, wird aufgerufen, wenn der Benutzer auf den pushbutton clickt).

3.1 Handhabung der Mausereignisse

3.1.1 Mausposition

Matlab entscheidet aufgrund der Benutzeraktion und der momentanen Mausposition, welche Callbacks auszuführen sind. Zur Unterscheidung der Mausposition am Bildschirm sind folgende Situationen von Bedeutung:

- Ist der Mauszeiger über einem Control platziert?
- Ist der Mauszeiger in der Nähe eines Controls (aktive Zone des Objekts)?
- Ist der Mauszeiger über oder in der Nähe eines anderen graphischen Objekts?
- Ist der Mauszeiger innerhalb eines Figurfensters aber nicht über oder in der Nähe eines Controls oder anderen graphischen Objekts?

Die auszuführenden Callbacks werden zu den gewünschten Objekten (Figur, Control oder andere graphische Objekte) als Eigenschaft zugeordnet. In der folgenden Tabelle ist aufgelistet, welche Eigenschaft (also welcher Callback) aufgrund der Kombination aus Mausposition und Benutzeraktion für ein Objekt aufgerufen wird.

Mausposition	Benutzeraktion	ausgelöste Eigenschaft
über einem Control oder Menu-Eintrag	Mausclick	`CallBack` Eigenschaft des Controls oder des Menus
in der Nähe eines Controls	Maustaste gedrückt	Eigenschaft `WindowButtonDownFcn` der Figur und `ButtonDownFcn` des Conrols
in der Nähe eines Controls	Maustaste loslassen	`WindowButtonUpFcn` der Figur
in der Nähe eines Controls	Maus bewegen	`WindowButtonMotionFcn` der Figur
über oder in der Nähe eines anderen graphischen Objekts (≠ Figur oder Control)	Maustaste gedrückt	`WindowButtonMotionFcn` der Figur und `ButtonDownFcn` des Objekts
über oder in der Nähe eines anderen graphischen Objekts (≠ Figur oder Control)	Maustaste loslassen	`WindowButtonUpFcn` der Figur
über oder in der Nähe eines anderen graphischen Objekts (≠ Figur oder Control)	Maus bewegen	`WindowButtonMotionFcn` der Figur

über keinem graphischen Objekt	Maustaste gedrückt	`WindowButtonDownFcn` der Figur
über keinem graphischen Objekt	Maustaste loslassen	`WindowButtonUpFcn` der Figur
über keinem graphischen Objekt	Maus bewegen	`WindowButtonMotion` der Figur

3.1.2 Drücken der Maustaste

Wenn der Benutzer die Maustaste innerhalb eines Figurfensters drückt, führt Matlab folgende Aktionen in angegebener Reihenfolge aus:

1) Matlab prüft, ob für die aktuelle Mausposition ein Objekt ausgewählt ist. Falls dies der Fall ist, setzt Matlab dieses Objekt in der Eigenschaft `'CurrentObject'` der Figur. Falls für die aktuelle Mausposition kein Objekt ausgewählt ist, führt die Eigenschaft `'CurrentObject'` der Figur ihren eigenen Handle.

2) Matlab setzt die Eigenschaften der Figur: `'CurrentPoint'` (s.u.) und `'Selection-Type'`.

3) Matlab platziert das ausgewählte Objekt als oberstes in der Liste (Stack) der momentan ausgewählten Objekte (vgl. Auswählen von Objekten mit der Maus).

4) Hat der Benutzer auf ein Control geclickt, wird dessen `'CallBack'` Eigenschaft ausgeführt. Schritte 5) und 6) entfallen.

5) Matlab führt die Callback-Eigenschaft `'WindowButtonDownFcn'` der Figur aus, sofern sie definiert wurde.

6) Matlab führt die Callback-Eigenschaft `'ButtonDownFcn'` des ausgewählten Objekts aus, sofern diese definiert wurde.

3.1.3 Loslassen der Maustaste

Lässt der Benutzer die Maustaste über dem gleichen Figurfenster wieder los, über dem er sie gedrückt hatte, wird für diese Figur folgendes ausgeführt:

1) Matlab aktualisiert die Eigenschaft `'CurrentPoint'` der Figur.

2) Matlab führt die Callback-Eigenschaft `'WindowButtonUpFcn'` der Figur aus.

3.1.4 Bewegen der Maus

Hat die Applikation nur ein Figurfenster oder wird die Maus bei mehreren Figurfenstern nur innerhalb einem bewegt, führt Matlab während der Mausbewegung folgende Aktionen aus:

1) Matlab aktualisiert die Eigenschaft `'CurrentPoint'` der Figur.

2) Matlab führt die Callback-Eigenschaft `'WindowButtonMotionFcn'` der Figur aus.

Erfolgt die Mausbewegung über mehrere Figurfenster, so ist zu unterscheiden, ob die Maustaste gedrückt ist oder nicht:

Wird die Maus einfach bewegt (ohne gedrückter Maustaste), so führt Matlab die obigen 2 Aktionen (`'CurrentPoint'`, `'WindowButtonMotionFcn'`) für jede Figur aus, in der sich der Mauszeiger gerade befindet.

Wird die Maus mit gedrückter Maustaste bewegt, so führt Matlab die obigen 2 Aktionen nur für die Figur aus, in der die Maus gedrückt wurde (auch wenn der Mauszeiger über diese Figur hinaus in eine andere Figur hinein bewegt wird).

Für alle Fälle unter 3.1.4 gilt zu beachten, dass `'CurrentPoint'` für eine Figur nur aktualisiert wird, wenn für sie auch die Eigenschaft `'WindowButtonMotionFcn'` definiert wurde.

3.2 Einige wichtige Eigenschaften

3.2.1 Eigenschaften der Figur

`'WindowButtonDownFcn'`

definiert die Callback-Eigenschaft, die Matlab aufruft, wenn der Benutzer die Maustaste drückt und sich der Mauszeiger innerhalb des Figurfensters an einer Stelle befindet, an der kein Control ist.

`'WindowButtonUpFcn'`

definiert die Callback-Eigenschaft einer Figur, die Matlab aufruft, wenn die Maustaste losgelassen wird. Es wird die Eigenschaft von derjenigen Figur ausgeführt, über der die Maus gedrückt wurde (auch wenn der Mauszeiger beim Loslassen der Maustaste über einer anderen Figur liegt).

`'WindowButtonMotionFcn'`

definiert die Callback-Eigenschaft, die Matlab ausführt, wenn der Benutzer die Maus innerhalb einer Figur bewegt.

`'CurrentObject'`

führt den Handle für das momentan in einer Figur ausgewählte Objekt (angeclicktes Control), bzw. den Handle der Figur selbst, falls kein Objekt im Figurfenster ausgewählt ist.

`'CurrentMenu'`

bezeichnet das bei gedrücketer Maustaste ausgewählte Menu oder Sub-Menu.

`'CurrentPoint'`

beinhaltet die x,y-Koordinaten der Mausposition, ausgedrückt in der Einheit, die in der Figureigenschaft Units definiert ist. Matlab setzt diese Eigenschaft, 1) wenn der Benutzer die Maustaste drückt, 2) wenn die Eigenschaft `WindowButtonUpFcn` definiert ist und der Benutzer die Maustaste loslässt und 3) wenn die Eigenschaft `WindowButtonMotionFcn` definiert ist und der Benutzer die Maus bewegt.

`'SelectionType'`

zeigt an, welche Maustaste gedrückt wurde, ob sie zusammen mit <shift> oder <ctrl> gedrückt wurde und ob es sich um einen einfachen oder Doppelclick handelt (vgl. Matlab Reference Guide).

`'Children'`

gibt momentane Stackreihenfolge aller in einer Figur platzierten Objekte an. Sie liefert eine Liste mit den Handles der Objekte zurück. Die Reihenfolge in der Liste entspricht der Stackreihenfolge der Objekte (vgl. 3.3.1 Stackreihenfolge).

3.2.2 Eigenschaften von Achsen

`'CurrentPoint'`

enthält die Koordinaten von zwei Punkten, die durch die Eigenschaft `'CurrentPoint'` definiert sind. Diese Eigenschaft der Achsen wird abgeleitet, wenn die Achse von der Eigenschaft `'CurrentPoint'` der Figur aufgefordert wird, die Figurkoordinaten in Achsenkoordinaten umzurechnen.

3.2.3 Eigenschaften von anderen Objekten

'CallBack'

diese Eigenschaft wird für Controls und Menus ausgeführt, wenn der Benutzer die entsprechende Aktion auslöst (Mausclick).

'ButtonDownFcn'

diese Eigenschaft führt Matlab aus, wenn der Benutzer auf die aktive Zone eines Objekts (s.u.) clickt.

3.3 Auswählen von Objekten mit der Maus

Wenn ein Objekt schwierig auszuwählen ist mit der Maus (wie z.B. ein Linienzug), oder wenn sich Objekte am Bildschirm überlappen, benutzt Matlab zwei Kriterien um festzustellen, welches Objekt zu selektieren ist, wenn der der Benutzer die Maustaste drückt:

- Die Stackreihenfolge der Objekte
- Die für die Objekte definierte aktive Zone .

Nur für das selektierte Objekt ruft Matlab dann den Callback auf. Wenn Callbacks für sich überlappende Objekte definiert werden, sollte der Programmierer berücksichtigen, ob einem späteren Benutzer klar ist, wie dieser die Objekte auswählen muss.

3.3.1 Stackreihenfolge

Wenn ein Benutzer die Maustaste an einer Stelle drückt, an der sich mehrere Objekte überlappen, bestimmt die Stackreihenfolge, welches Objekt selektiert wird. Zu Programmstart ist die Stackreihenfolge die Reihenfolge, in der die überlappenden Objekte durch den Programmcode erzeugt wurden. Die zuletzt erzeugten Objekte liegen in höherer Position im Stack als die früher erzeugten. Wird die Maustaste gedrückt, wenn der Mauszeiger über einem Objekt liegt, wandert dieses selektierte Objekt im Stack an die höchste Stelle.

Somit entspricht die Stackreihenfolge nicht unbedingt der sichtbaren Überlappung der Objekte am Bildschirm. Objekt, die quasi dreidimensional näher am Betrachter liegen, müssen nicht höher in der Stackreihenfolge stehen als die anderen Objekte, die sie z. T. zudecken.

Die momentane Stackreihenfolge aller in einer Figur platzierten Objekte kann über die Eigenschaft 'Childern' der Figur abgefragt werden. Sie liefert eine Liste mit den Handles der Objekte zurück. Die Reihenfolge in der Liste entspricht der Stackreihenfolge.

3.3.2 Aktive Zone

Die aktive Zone eines Objekts ist der Bereich, in dem es mit der Maus selektiert werden kann. Die aktive Zone umfasst die Ausmasse des Objekts selbst. Sie kann aber zusätzlich noch einen selektierbaren Rand beinhalten (z.B. bei sehr kleinen oder sehr schmalen Objekten wie z.B. Linienzüge).

Linienobjekte

Die aktive Zone von Linienobjekten besteht aus der Linie selbst und einem 5 Pixel breiten Rand rings herum um die Linie. Der selektierbare Bereich einer Linie ist also 10 Pixel + Linienstärke breit.

Achsenobjekte

Die aktive Zone von Achsenobjekten besteht aus dem (mit Einheiten beschrifteten) Rechteck, das die Graphik begrenzt, sowie aus dem Bereich aussen herum, wo die Labels und der Titel stehen. Diese aktive Zone beinhaltet nicht Objekte oder Controls, die innerhalb des Achsenrechtecks liegen.

Oberflächen- und Textobjekte

Die aktive Zone von Oberflächen- und Textobjekten ist das kleinste Reckteck aus waagrechten und senkrechten Linien, das das Objekt einschliesst.

Controls

Für Controls können zweierlei Callbacks definiert werden: `CallBack` und `ButtonDownFcn`. Da beide Eigenschaften durch Drücken der Maustaste ausgelöst werden, unterscheidet Matlab zwei verschiedene Bereiche, denen die Callbacks zugeordnet werden:

- Wird die Maustaste gedrückt, wenn der Mauszeiger über dem Control steht, führt Matlab für das Control die Eigenschaft `CallBack` aus.
- Die aktive Zone des Controls umgibt diesen mit einem 5 Pixel breiten Rand. Wird die Maustaste gedrückt, wenn sich der Mauszeiger in diesem Rahmen befindet, wird die Eigenschaft `ButtonDownFcn` für das Contol ausglöst. Ist zusätzlich die `WindowButtonDownFcn` für die Figur definiert, so wird diese zuerst ausgeführt.

3.4 Bestimmung der aktuellen Mausposition (CurrentPoint)

Matlab führt die aktuelle Mausposition in der Eigenschaft `'CurrentPoint'` der Figur bei folgenden Benutzeraktionen nach:

- wenn der Benutzer die Maustaste drückt, unabhängig davon, ob für die Figur die Eigenschaft `'WindowButtonDownFcn'` definiert ist,
- wenn der Benutzer die Maustaste wieder loslässt und für die Figur die Eigenschaft `'WindowButtonUpFcn'` definiert ist.
- wenn der Benutzer die Maus bewegt und für die Figur die Eigenschaft `'WindowButtonMotionFcn'` definiert ist.

Die Eigenschaft `'CurrentPoint'` der Figur enthält die x,y-Koordinaten des Mauszeigers von der linken unteren Ecke des Figurfensters gerechnet. Die Eigenschaft `'CurrentPoint'` der Achsenobjekte enthält eine 2*3-Matrix mit einem Paar von Koordinaten in dem Achsen-Datenraum.

Der Bildschirm zeigt die 3-D-Graphiken in der 2-D-Ebene. Um den Verlust der dritten Dimension auszugleichen, gibt Matlab in der 2*3-Matrix die beiden 3-D-Koordinaten der Endpunkte zurück, welche eine Linie von vorn nach hinten im 3-D-Raum der Achsen bilden und zwar in Draufsicht auf die Bildschirmfläche. Der Raum der Achsen ist durch seine x,y,z-Achsenbegrenzugn gegeben.

Der Wert von `'CurrentPoint'` der Achse hat die Form:

```
[Xback   Yback   Zback;
 Xfront  Yfront  Zfront].
```

Die Koordinaten sind in dem Datenraum der jeweiligen Achse angegeben, d.h. in der Einheit, in der auch die Daten für die jeweilige Achse geplottet werden. Der Mauszeiger muss sich nicht innerhalb der Achsen oder innerhalb der Figur befinden. Die Daten werden auch ausserhalb in der richtigen Einheit des Ojekts angegeben.

Beispielanwendung zur Bestimmung der aktuellen Mausposition

Dieses Beispiel zeigt, wie ein pushbutton durch die Mausbewegung verschoben werden kann: Um einen vorhandenen pushbutton an einer anderen Stelle zu platzieren, soll zuerst die Maus an irgendeiner Position (ausser direkt über dem pushbutton) gedrückt werden und mit gedrückter Maustaste zu dem neuen Platzierungspunkt gezogen werden. Dort ist die Maustaste loszulassen, und der pushbutton wird an dieser Stelle neu gezeichnet. Der Code hierfür sieht folgendermassen aus:

```
fig = figure('Color', 'w');
pos_vec = [10 10 75 25];
pb = uicontrol(fig,'Style','push', 'Position',pos_vec);
wbd_cb = ['set(fig, ''WindowButtonMotionFcn'', ['...
 ' ''pv = get(fig, ''''CurrentPoint'''');'' ',...
 ' ''pos_vec(1) = pv(1);'' ',...
```

```
 ' ''pos_vec(2) = pv(2);''])'];
cur_pos = 'set(pb, ''Pos'', pos_vec)';
set(fig,'WindowButtonDownFcn', wbd_cb)
set(fig, 'WindowButtonUpFcn', [...
 'set(fig, ''WindowButtonMotionFcn'', '' ''),',cur_pos])
```

In dieser Implementation zeichnet Matlab den pushbutton erst, wenn der Benutzer die Maustaste innerhalb des Figurfensters loslässt (`'WindowButtonUpFcn'`). Die Mausposition in der Eigenschaft `'CurrentPoint'` wird dagegen ständig aufdatiert, während der Benutzer die Maus mit gedrückter Maustaste bewegt.

Das Beispiel zeigt die drei Callbacks, die mit den Benutzeraktionen Maustaste drücken, Maus bewegen und Maustaste loslassen ausgelöst werden:

- `'WindowButtonDownFcn'` setzt den Callback für die Eigenschaft `'WindowButtonMotionFcn'`. Hätte man die `'WindowButtonMotionFcn'` für die Figur bei deren Erzeugung fest definiert, würde jede Mausbewegung den pushbutton verschieben. Statt dessen löst erst das Drücken der Maustaste die Definition des Callbacks aus, das die Knopfbewegung veranlasst.
- Die 'WindowButtonMotionFcn' führt in der Eigenschaft `'CurrentPoint'` die Position der Maus nach. die x,y-Koordinaten der Mausposition ersetzen die ersten beiden Einträge im Positionsvektor des pushbuttons.
- Die `'WindowButtonUpFcn'` löscht den Callback aus der `'WindowButtonMotionFcn'` wieder und platziert den pushbutton an neuer Stelle.

3.5 Unterbrechen (Interrupting) von Callbacks

Wird von Matlab ein Callback aufgerufen, so wird dieser normalerweise an einem Stück ohne Unterbrechung abgearbeitet. Es gibt jedoch bestimmte vorrangige Ereignisse, die den Callback unterbrechen (z.B. der Benutzer clickt die Maustaste erneut oder die Maus wird während der Callbackausführung verschoben). Um die Unterbrechung der Callback-Ausführung zu verhindern, ist für das Objekt die Eigenschaft `'Interruptable', 'no'` zu setzen (default: `'Interruptable', 'yes'`). Diese Eigenschaft kann für jedes Objekt gesetzt werden, das auch die Eigenschaft `'CallBack'` besitzen kann - also für alle graphischen Objekte. Ist `'Interruptable', 'yes'` gesetzt, so kann die Callback-Anweisung dieses Objekts durch die Callback-Anweisung eines anderen Objekts unterbrochen werden.

Callbacks werden ohne Unterbruch ausgeführt bis bestimmte Matlabbefehle auftreten. Dann beginnt Matlab die hängigen Ereignisse, die in der Ereigniswarteschlange stehen, abzuarbeiten.

3.5.1 Ereignisse und die Ereigniswarteschlange

Matlabbefehle, die numerische Berechnungen ausführen oder die Eigenschaften von Graphikobjekten setzen, werden sofort ausgeführt. Befehle oder Aktionen, die Eingänge oder Ausgänge zu einem Figurfenster beinhalten, generieren ein Ereignis. Diese Ereignisse schliessen auch Mausaktionen, und Redraw-Anweisungen für die Graphiken ein.

Matlab speichert diese Ereignisse in einer Ereigniswarteschlange bis sie verarbeitet werden können. Die Ereignisschlange wird überprüft, wenn Matlab folgende Befehle während eines Callbacks für ein Objekt bearbeitet:

- `drawnow`
- `figure` (einschliesslich der Befehle gcf und gca)
- `getframe`
- `pause`

Wenn Matlab eine dieser Anweisungen während einer Callback-Ausführung antrifft, unterbricht es den Callback und bearbeitet zuerst die Ereignisse in der Ereigniswarteschlange. Dies ist abhängig von der Eigenschaft `'Interruptable'` des Objekts, dessen Callback gerade ausgeführt wird:

- Ereignisse der Warteschlange, für die ein Callback auszuführen ist, - wie z.B. Drücken der Maustaste, Loslassen der Maustaste, Bewegung der Maus - werden nur behandelt, wenn das zu unterbrechende Objekt die Eigenschaft `'Interruptable'. 'yes'` gesetzt hat.
- Um Redraw-Ereignisse auszuführen, wird der laufende Callback auch unterbrochen, wenn die Eigenschaft `'Interruptable', 'no'` gesetzt ist. Um selbst den Redraw während eines Callbacks zu unterbinden, muss der Befehl

  ```
  drawnow discard
  ```

  verwendet werden. Wenn Matlab auf einen `drawnow` mit der Option `discard` stösst, löscht es alle Ereignisse in der Warteschlange

3.5.2 Ausführung eines Callbacks

Wird ein Callback von Matlab ausgeführt, laufen folgende Schritte ab:

1. Wird einer der Befehle `drawnow`, `figure`, `getframe` oder `pause` erkannt, wird der laufende Callback gestoppt und und die Ereigniswarteschlange bearbeitet.
2. Falls es sich um den Befehl `drawnow` mit der Option `discard` handelt, werden alle hängigen Ereignisse in der Warteschlange gelöscht und es wird mit 4. weitergefahren.
3. Ist das oberste Ereignis in der Warteschlange ein Redraw, so wird das Neuzeichnen ausgelöst und zu dem nächsten Ereignis in der Warteschlange weitergegangen.

   Muss für ein Ereignis in der Warteschlange ein weiterer Callback aufgerufen werden, so wird überprüft, ob der gerade unterbrochene Callback `'Interruptable', 'yes'` hat. Nur in diesem Fall wird der weitere Callback ausgeführt. Beinhaltet auch dieser Callback einen der Befehle `drawnow`, `figure`, `getframe` oder `pause`, so wiederholt Matlab die Schritte 1. bis 4. für die verbleibenden Ereigneisse in der Warteschlange.

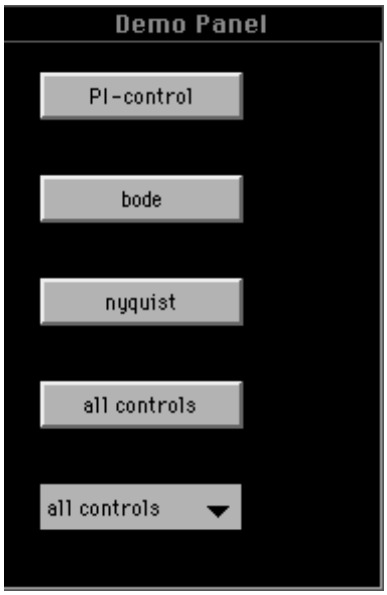
   Hat der unterbrochene Callback `'Interruptable', 'no'`, so wird das anstehende Ereignis übergangen und das nächste Ereignis in der Warteschlange bearbeitet.
4. Ist die Warteliste leer, so wird mit der Abarbeitung des unterbrochenen Callbacks fortgefahren.

Wenn Matlab einen Callback unterbricht, um ein Ereignis zu bearbeiten, so wird der momentante Zustand aller Objekte nicht gespeichert (das wären `'CurrentFigure'`, `'CurrentAxis'`, die Objekteigenschaften und auch die Variablen im Workspace). Somit ist zu beachten, dass die Ereignisabarbeitung den Zustand verändern kann und bei Wiederaufnahme des unterbrochenen Callbacks andere Verhältnisse herrschen. D.h. jeder Callback ist selbst verantwortlich, die Daten zu erhalten, die er später wieder braucht.

Alle Mausbewegungen ergeben einen Eintrag in der Ereigniswarteschlange. Wenn Matlab die Warteschange bearbeitet und auf ein Mausbewegungsereignis stösst, bearbeitet es nur das neueste Mausbewegungsereignis und löscht alle alten.

Beispiele zur Gestaltung graphischer Oberflächen mit der Student Edition von Matlab

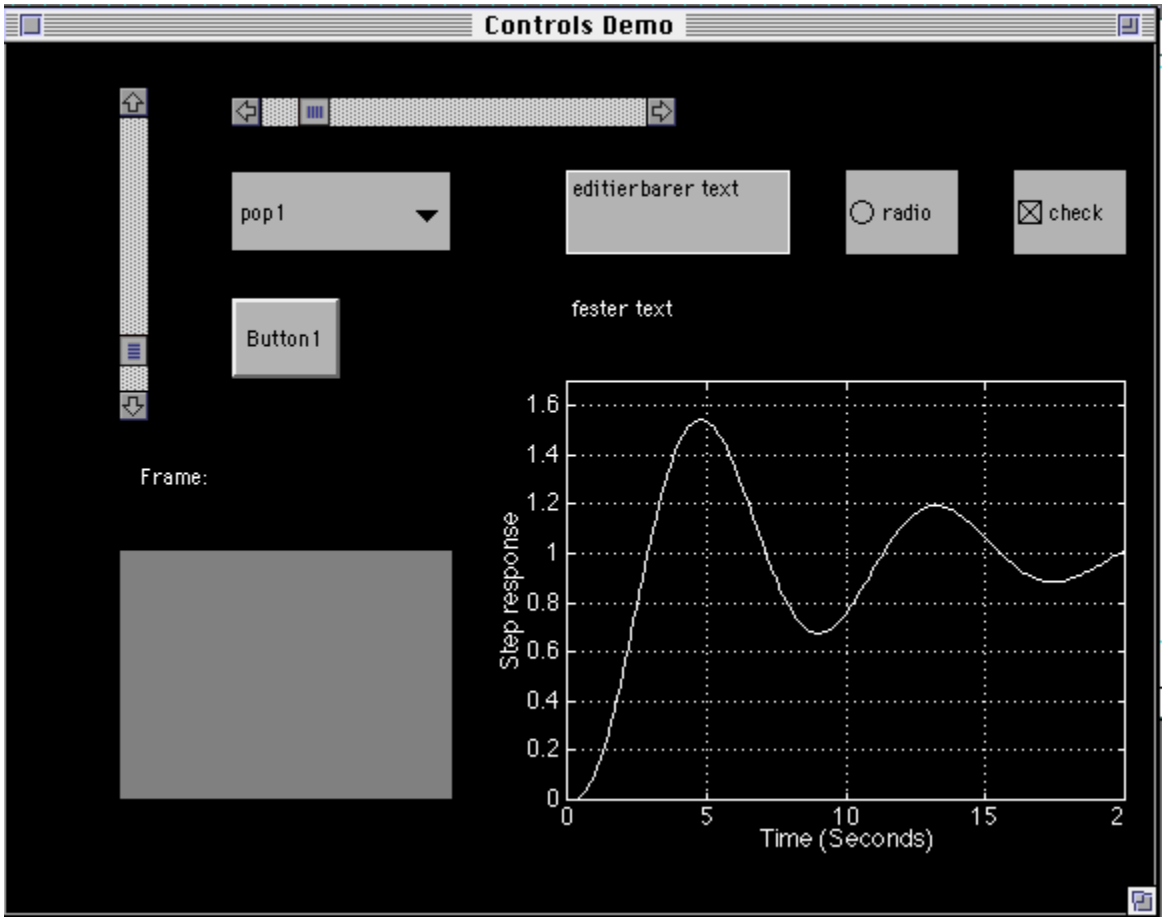
Es werden jeweils ein Bildschirm und das dazugehörende File angegebene.



Das Hauptpanel

```
% file demogui.m
% load all demonstrations about programming the graphical user interfaces

global menuselecion

% first create a new figure
oldFigNumber= watchon;
figNumber=figure;

set(gcf, ...
        'NumberTitle','off', ...
        'Name','Demo Panel', ...
            'backingstore','off',...
            'Units','normalized');

set(gcf, 'Position', [0.01 0.7 0.15 0.3]);
% then add controls

uicontrol('Style','Pushbutton', ...
        'Position',[0.1 0.25 0.6 0.1], ...
        'Units','normalized',...
        'Callback','ShowAllControls;','String','all controls');
```

```
uicontrol('Style','Pushbutton', ...
        'Position',[0.1 0.45 0.6 0.1], ...
        'Units','normalized',...
        'Callback','nyquistgui;','String','nyquist');


uicontrol('Style','Pushbutton', ...
        'Position',[0.1 0.65 0.6 0.1], ...
        'Units','normalized',...
        'Callback','bodegui;','String','bode');


uicontrol('Style','Pushbutton', ...
        'Position',[0.1 0.85 0.6 0.1], ...
        'Units','normalized',...
        'Callback','wscontrol;','String','PI-control');

menuselection=uicontrol('Style','Popup','Position',[0.1 0.05 0.6 0.1],...
        'Units','normalized', 'String', 'all controls|nyquist|bode|PI-control',
...
        'Callback','demoguiact');
```

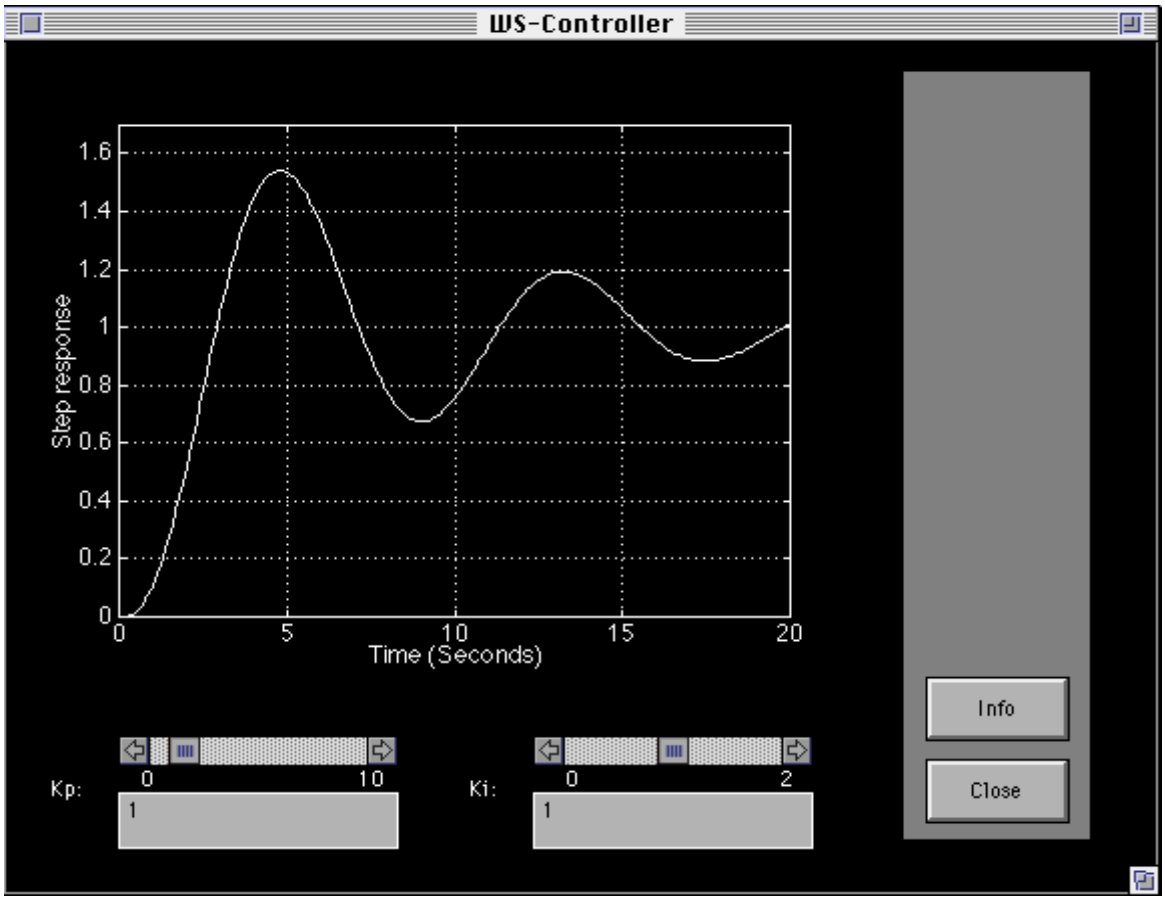


Dieses Beispiel zeigt alle graphisachen Möglichkeiten ohne Funktionalität

```
% A Matlab file to show all control elements
% first create a new fidure
oldFigNumber= watchon;
figNumber=figure;
set(gcf, ...
        'NumberTitle','off', ...
        'Name','Controls Demo', ...
            'backingstore','off',...
            'Units','normalized');
% then add controls
% a frame
frmPos=[0.1 0.1 0.3 0.3];
h=uicontrol( ...
        'Style','frame', ...
        'Units','normalized', 'String', 'frame', ...
        'Position',frmPos, ...
        'BackgroundColor',[0.5 0.5 0.5]);
% and a text field
```

```
uicontrol('style','text','Position',[.05 .4 .2 .1],...
        'Units','normalized','BackgroundColor','black',...
        'ForegroundColor','white','String','Frame:');

% a pushbutton
button1=uicontrol('Style','Pushbutton', ...
        'Position',[0.2 0.6 0.1 0.1], ...
        'Units','normalized',...
        'Callback','disp(''button1'')','String','Button1');

% two sliders
slider1=uicontrol('Style','slider','Position',[0.2 0.9 0.4 .04],...
        'Units','normalized','Value',1,'Max',10,'Min',0,...
        'Callback','disp(''slider1'')');
slider2=uicontrol('Style','slider','Position',[0.1 0.55 0.03 .4],...
        'Units','normalized','Value',1,'Max',10,'Min',0,...
        'Callback','disp(''slider2'')');

% a checkbox
checkbox1=uicontrol('Style','checkbox','Position',[0.9 0.75 0.1 0.1],...
        'Units','normalized','Value',1,'String', 'check', ...
        'Callback','disp(''checkbox1'')');

% popupmenu
popup1=uicontrol('Style','Popup','Position',[0.2 0.75 0.2 0.1],...
        'Units','normalized', 'String', 'pop1|pop2|pop3|pop4', ...
        'Callback','disp(''Popup1'')');

% a step response
    % Create initial system
    kp = 1;   ki= 1;   T=0:0.1:20; poles=[-1 -1 -1];
      nPlant = [1];      dPlant = poly(poles);
      nContr = [kp ki];  dContr = [1 0];
      [nSyso, dSyso] = series(nPlant, dPlant, nContr, dContr);
      [nSysc, dSysc] = cloop(nSyso, dSyso, -1);
    [Ys,Xs] = step(nSysc, dSysc,T);  % step response (closed_loop);
    ax_step=axes('Position',[0.5 0.1 0.5 0.5],'XLim',[0 max(T)],...
                                          'YLim',[min(Ys) 1.1*max(Ys)]);
```

```
    step_line=plot(T,Ys,'w');
    axis([0 max(T) min(Ys) 1.1*max(Ys)]);
    grid;
    ylabel('Step response');
    xlabel('Time (Seconds)');

% a text field
    uicontrol('style','text','Position',[.5 .6 .1 .1 ],...
        'Units','normalized','BackgroundColor','black',...
        'ForegroundColor','white','String','fester text');

% an edit field
     uicontrol('style','edit','Position',[.5 .75 .2 .1 ],...
        'Units','normalized','String','editierbarer text', ...
            'Callback','disp(''edit text1'')');

% a radio button
rbutton1=uicontrol('Style','radio','Position',[0.75 0.75 0.1 0.1],...
        'Units','normalized', 'String', 'radio',...
        'Callback','disp(''Radio button 1'')');
```



Trainingsprogramm für PI-Regler

```
function WSControl(action,in1);
%  WSControl Interactive control demo: plant of 3d order with PI-controller
%  The transfer function of plant:
%                                1
%          H(s) =   -------------
%                       (s+1)^3
%     Demonstrates MATLAB's graphic user interface using Handle Graphics
%   while illustrating PI-Controller properties.

%     Author: X. Qiu, W. S.
%     10/5/95  20/9/95
%

%     possible actions:
%       'start'
%       'redraw'     - internal
%       'setkp'      - in1=1 ==> from slider, in1=2 ==> from edit text
%       'setki'      - in1=1 ==> from slider, in1=2 ==> from edit text
```

```
%       'info'
%       'done'

if nargin<1,
    action='start';
end;

global WSControl_DAT T

if strcmp(action,'start'),

    %====================================
    % Graphics initialization
    oldFigNumber = watchon;
    figNumber = figure;
    set(gcf, ...
        'NumberTitle','off', ...
        'Name','WS-Controller', ...
      'backingstore','off',...
      'Units','normalized');

    %====================================
    % Information for all buttons
    top=0.95;
    bottom=0.05;
    left=0.82;
    yInitLabelPos=0.90;
    btnWid = 0.13;
    btnHt=0.08;
    % Spacing between the label and the button for the same command
    btnOffset=0.02;
    % Spacing between the button and the next command's label
    spacing=0.02;
    %bottom=bottom+spacing;

    %====================================
    % The CONSOLE frame
    frmBorder=0.02;
```

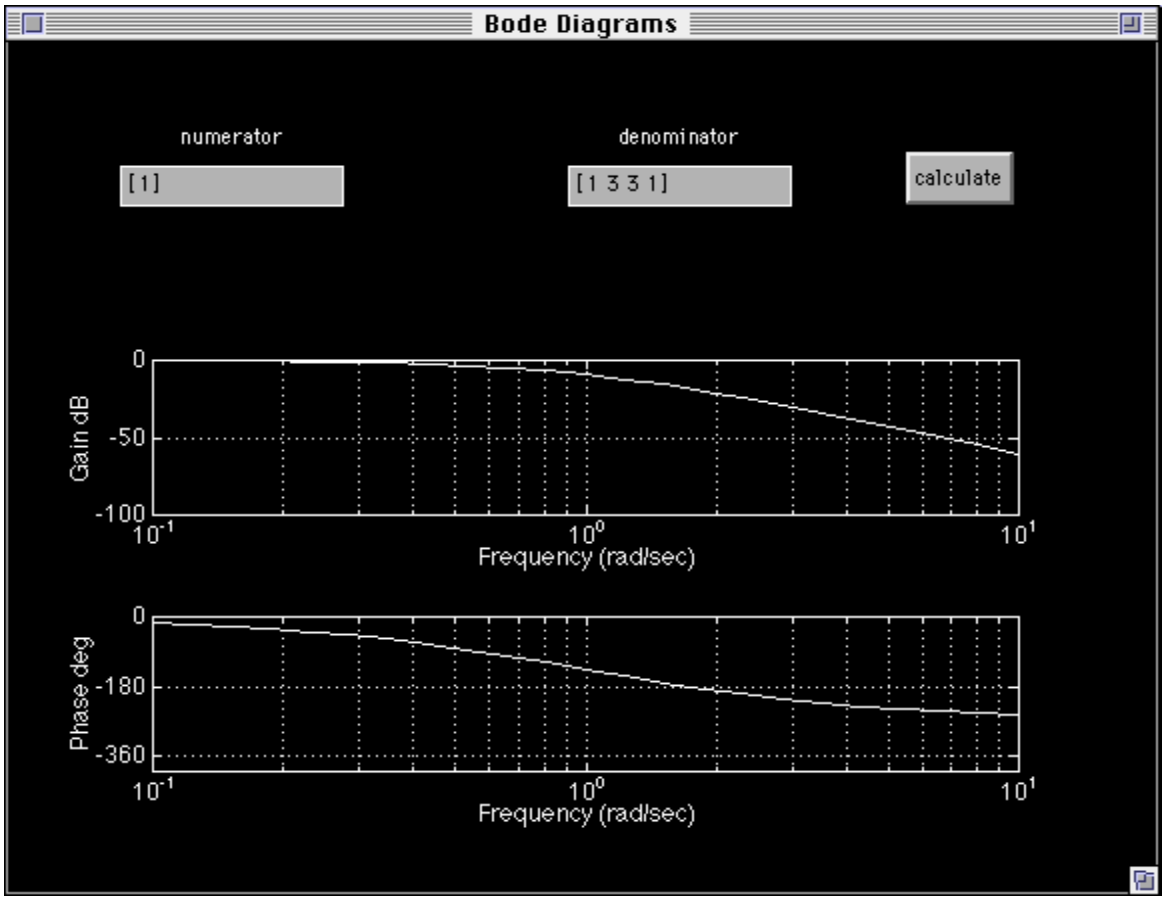
```
 yPos=0.02;
 frmPos=[left-frmBorder bottom-frmBorder btnWid+2*frmBorder ...
       0.9+2*frmBorder];
 h=uicontrol( ...
     'Style','frame', ...
     'Units','normalized', ...
     'Position',frmPos, ...
     'BackgroundColor',[0.5 0.5 0.5]);

 %====================================
 % The INFO button
 uicontrol( ...
     'Style','push', ...
     'Units','normalized', ...
     'Position',[left bottom+btnHt+spacing  btnWid btnHt], ...
     'String','Info', ...
     'Callback','WSControl(''info'')');

%=======================================
% The CLOSE button
 done_button=uicontrol('Style','Pushbutton', ...
     'Position',[left bottom btnWid btnHt], ...
     'Units','normalized',...
     'Callback','WSControl(''done'')','String','Close');
%====================================

 % Create initial system
 min_kp = 0;  max_kp = 10;
   min_ki = 0;  max_ki = 2;
 kp = 1;   ki= 1; T=0:0.1:20;

   % calculate the closed-loop transfer function
   [nSysc, dSysc] = TFC([-1; -1; -1], kp, ki);
 [Ys,Xs] = step(nSysc, dSysc,T);  % step response (closed_loop);

 kp_text=uicontrol('Style','text','Position',[.03 .04 .05 .07],...
     'Units','normalized','BackgroundColor','black',...
     'ForegroundColor','white','String','Kp:');
```

```
uicontrol('style','text','pos',[.10 .07 .05 .05],...
    'Units','normalized','BackgroundColor','black',...
    'ForegroundColor','white','String',num2str(min_kp));

uicontrol('style','text','pos',[.30 .07 .05 .05 ],...
    'Units','normalized','BackgroundColor','black',...
    'ForegroundColor','white','String',num2str(max_kp));

kp_field=uicontrol('Style','edit','Position',[.10 .02 .25 .07],...
    'Units','normalized','String',num2str(kp),...
    'CallBack','WSControl(''setkp'',2); WSControl(''redraw'');');

kp_slider=uicontrol('Style','slider','Position',[.10 .12 .25 .04],...
    'Units','normalized','Value',kp,'Max',max_kp,'Min',min_kp,...
    'Callback','WSControl(''setkp'',1); WSControl(''redraw'');');

ki_text=uicontrol('Style','text','Position',[.40 .04 .05 .07],...
    'Units','normalized','BackgroundColor','black',...
    'ForegroundColor','white','String','Ki:');

uicontrol('style','text','pos',[.48 .07 .05 .05],...
    'Units','normalized','BackgroundColor','black',...
    'ForegroundColor','white','String',num2str(min_ki));

uicontrol('style','text','pos',[.67 .07 .05 .05 ],...
    'Units','normalized','BackgroundColor','black',...
    'ForegroundColor','white','String',num2str(max_ki));

ki_field=uicontrol('Style','edit','Position',[.47 .02 .25 .07],...
    'Units','normalized','String',num2str(ki),...
    'CallBack','WSControl(''setki'',2); WSControl(''redraw'');');

ki_slider=uicontrol('Style','slider','Position',[.47 .12 .25 .04],...
    'Units','normalized','Value',ki,'Max',max_ki,'Min',min_ki,...
    'Callback','WSControl(''setki'',1); WSControl(''redraw'');');

% step response plot
```

```
    ax_step=axes('Position',[.1 .3 .6 .6],'XLim',[0 max(T)],...
                                          'YLim',[min(Ys) 1.1*max(Ys)]);
    step_line=plot(T,Ys,'w');
    axis([0 max(T) min(Ys) 1.1*max(Ys)]);
    grid;
    ylabel('Step response');
    xlabel('Time (Seconds)');


    drawnow;


    WSControl_DAT = ...
      [-1; -1; -1; kp; min_kp; max_kp; ki; min_ki; max_ki; kp_field; kp_slider;
ki_field; ki_slider; ax_step; step_line];
    % 1   2   3   4       5       6   7       8        9         10          11
12         13        14         15
    watchoff(oldFigNumber);


      elseif strcmp(action,'setkp'),
    if (in1==1),    % set from slider
        WSControl_DAT(4)=get(WSControl_DAT(11),'value');
    else  % set from edit text
        min_kp=WSControl_DAT(5);
        max_kp=WSControl_DAT(6);
        kp=str2num(get(WSControl_DAT(10),'string'));
        if isempty(kp),    % handle non-numeric input into field
            set(WSControl_DAT(10),'string',num2str(WSControl_DAT(4)));
        else
            if (kp>max_kp),
                kp=max_kp;
            end;
            if (kp<min_kp),
               kp=min_kp;
            end;
            WSControl_DAT(4)=kp;
        end
    end


      elseif strcmp(action,'setki'),
    if (in1==1),    % set from slider
```

```
        WSControl_DAT(7)=get(WSControl_DAT(13),'value');
    else  % set from edit text
        min_ki=WSControl_DAT(8);
        max_ki=WSControl_DAT(9);
        ki=str2num(get(WSControl_DAT(12),'string'));
        if isempty(ki),    % handle non-numeric input into field
            set(WSControl_DAT(12),'string',num2str(WSControl_DAT(7)));
        else
            if (ki>max_ki),
                ki=max_ki;
            end;
            if (ki<min_ki),
               ki=min_ki;
            end;
            WSControl_DAT(7)=ki;
        end
    end

elseif strcmp(action,'redraw'),
      kp = WSControl_DAT(4);
      ki = WSControl_DAT(7);
      set(WSControl_DAT(10),'string',num2str(kp));
    set(WSControl_DAT(11),'value',kp);
      set(WSControl_DAT(12),'string',num2str(ki));
    set(WSControl_DAT(13),'value',ki);

      [nSysc, dSysc] = TFC([-1; -1; -1], kp, ki);
    [Ys,Xs] = step(nSysc, dSysc, T);

    set(WSControl_DAT(14),'XLim',[0 max(T)],...
                                    'YLim',[min(Ys) 1.1*max(Ys)]);
    set(WSControl_DAT(15),'YData',Ys,'XData',T);

    drawnow;

      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      elseif strcmp(action,'info'),
    ttlStr='Step Responses with P-I controller';
```

```
    hlpStr1= ...
        ['                                                  '
         ' We are using a P-I Controller on a third order  '
         ' plant. The step responses can be changed by     '
         ' changing kp and ki                              '];

    hlpStr2= ...
        ['                                                   '
         ' There are two ways of changing the parameters    '
         ' by either using the scroll bars                  '
         ' or by using the editing fields connected to them '];

     hlpStr3= 'File name: WSControl.m';

     helpfun(ttlStr, hlpStr1, hlpStr2, hlpStr3);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      elseif strcmp(action,'done'),
    close(gcf);
    clear global WSControl_DAT
end
```



Programm für die Berechnung und Darstellung von Bode-Diagrammen

```
function bodegui(action);

global numbode numtbode denbode dentbode b1bode ax_bode

if nargin==0, action='initialize'; end;

% playground for uicontrols

if strcmp(action, 'initialize'),
    % Graphics initialization
    oldFigNumber = watchon;
    figNumber = figure;
    set(gcf, ...
        'NumberTitle','off', ...
        'Name','Bode Diagrams', ...
      'backingstore','off',...
      'Units','normalized');
      % create controls
```

```
      clf;
      h=gcf; % get current figure handle

      % install two edit fields with text
    numbode = uicontrol(h,'Style','edit','Position',[.1 .8 .2 .05 ],...
      'Units','normalized','String','[1]', ...
          'Callback','disp(''edit text1'')');
      numtbode= uicontrol(h,'Style','text','Position',[.1 .85 .2 .05 ],...
      'Units','normalized','BackgroundColor','black',...
      'ForegroundColor','white','String','numerator');


      denbode = uicontrol(h,'Style','edit','Position',[.5 .8 .2 .05 ],...
      'Units','normalized','String','[1 3 3 1]', ...
          'Callback','disp(''edit text1'')');
      dentbode= uicontrol(h,'Style','text','Position',[.5 .85 .2 .05 ],...
      'Units','normalized','BackgroundColor','black',...
      'ForegroundColor','white','String','denominator');

      b1bode= uicontrol(h,'Style','Pushbutton', ...
      'Position',[0.8 0.8 0.1 0.07], ...
      'Units','normalized',...
      'Callback','bodegui(''calculate'')','String','calculate');

      % ax_bode=axes('Position',[.1 .1 .6 .6]);

elseif strcmp(action,'calculate'),
      % reset(ax_bode);
      numerator=get(numbode,'String');
      denominator=get(denbode,'String');
      numerator, denominator,
      % [mag,phase,w]=bode(str2num(numerator),str2num(denominator));
      % plot(re,im);
      bode2(str2num(numerator),str2num(denominator));
    drawnow;

end;
```


Nyquist Diagrams