Erstellen von grafischen Bedienoberflächen mit MATLAB

1. Einführung

Wie man beim Programmieren mit PEARL oder C die MATLAB-Engine nutzen kann, beispielsweise um für bestimmte Berechnungen geeignete MATLAB-Funktionen aufzurufen oder um die vielfältigen Grafik-Möglichkeiten von MATLAB für die Darstellung von Ergebnissen zu verwenden, wurde bereits in [2] für das Arbeiten unter Windows beschrieben. Über die MATLAB-Engine kann man aber auch MATLAB-GUI´s (Graphical User Interfaces) aufrufen und so die Mensch-Maschine-Kommunikation abwickeln. Das ist von besonderem Interesse, weil MATLAB über sehr leistungsfähige Werkzeuge zur Entwicklung solcher GUI´s verfügt und komfortable Bedienoberflächen sich daher ohne großen Aufwand erstellen lassen. MATLAB-GUI´s sind interaktive Dialogboxen bzw. MATLAB-Figures, die unterschiedliche grafische Objekte (Komponenten des GUI), wie Text- und Grafikausgaben, Buttons verschiedener Art usw. enthalten können. Ein Beispiel für solch ein GUI zeigt Abbildung 1.
Die Erstellung von GUI´s unter MATLAB und deren Nutzung durch ein PEARL-Programm wird im Folgenden beschrieben.

2. Beispiel „Messwertfilterung“

Als Beispiel für die folgenden Darlegungen soll ein einfaches Programm zur Messwerterfassung und -filterung dienen. Die Werte einer Messgröße x werden durch das PEARL-Programm „Messwertfilterung“ zyklisch erfasst und in einem festgelegten Zeitintervall als Messwertvektor an MATLAB übergeben. Unter MATLAB läuft auch die grafische Bedienoberfläche (GUI) des Programms (Abb. 1), die vor der Messwertübertragung vom PEARL-Programm (einmalig) aktiviert wird.

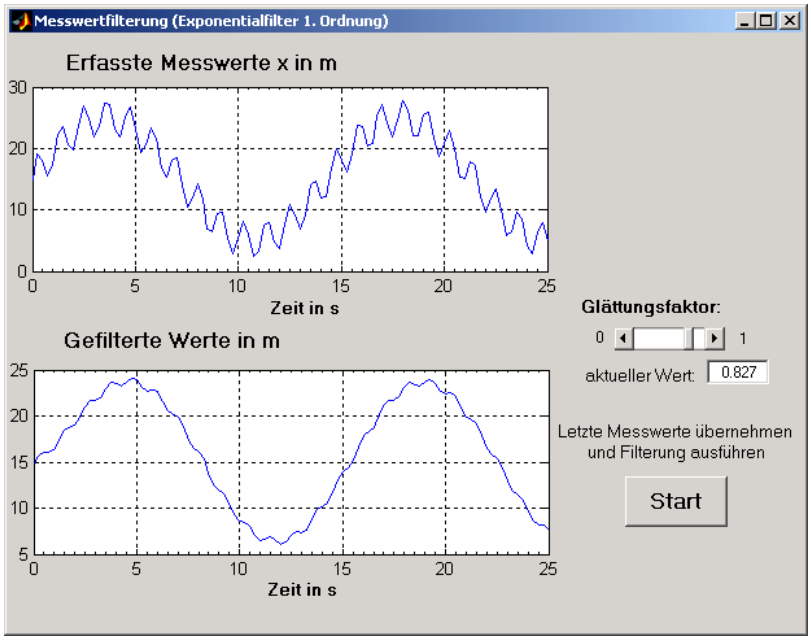

Abbildung 1: GUI des Programms „Messwertfilterung“

Der Anwender hat die Möglichkeit, über einen Schieber oder über die darunter befindliche editierbare Box einen gewünschten Filterfaktor (Glättungsfaktor) vorzugeben und danach mit Hilfe des Start-Buttons die Übernahme der Messwerte durch das GUI und die Berechnung der gefilterten Werte bzw. die Anzeige der Ergebnisse zu veranlassen.

Jedes MATLAB-GUI wird durch zwei Files beschrieben, im vorliegenden Beispiel durch die Files *messwertfilterung.fig* und *messwertfilterung.m*. Im Figure-File *messwertfilterung.fig* ist das Erscheinungsbild des grafischen Interfaces festgelegt, im so genannten Application-M-File *messwertfilterung.m* dessen Funktionalität.

3. Erstellen des GUI-Layouts

Das Erstellen des Layouts eines GUI unterstützt MATLAB durch die Entwicklungsumgebung GUIDE (*GUI Development Environment*). Sie umfasst einen grafischen Layout-Editor, den Editor für die Komponenteneigenschaften (*Property Inspector*), den Objekt-Browser für das Anzeigen einer Liste der Komponenten, den Menü-Editor und einen Editor für die Festlegung der Reihenfolge der Komponentenauswahl beim Drücken der Tab-Taste (*Tab Order Editor*). Nach dem Aufruf der GUIDE erscheint auf dem Bildschirm der Layout-Editor mit einer Palette der verfügbaren GUI-Komponententypen und einer leeren GUI-Figure. Aus dem angezeigten Vorrat von Komponenten kann man nun die gewünschten mit dem Mauszeiger auswählen bzw. anklicken und auf der GUI-Figure platzieren. Mit der Maus lässt sich auch die Größe der Komponenten verändern.
Die Möglichkeiten der Gestaltung eines GUI demonstriert Abbildung 2. Diese zeigt das Editorfenster mit einer GUI-Figure, die von jedem Typ eine Komponente enthält. Die Namen der Komponenten bzw. Objekte vergibt der Editor automatisch, sie sollten aber zweckmäßigerweise vom Programmierer durch problembezogene Bezeichnungen ersetzt werden

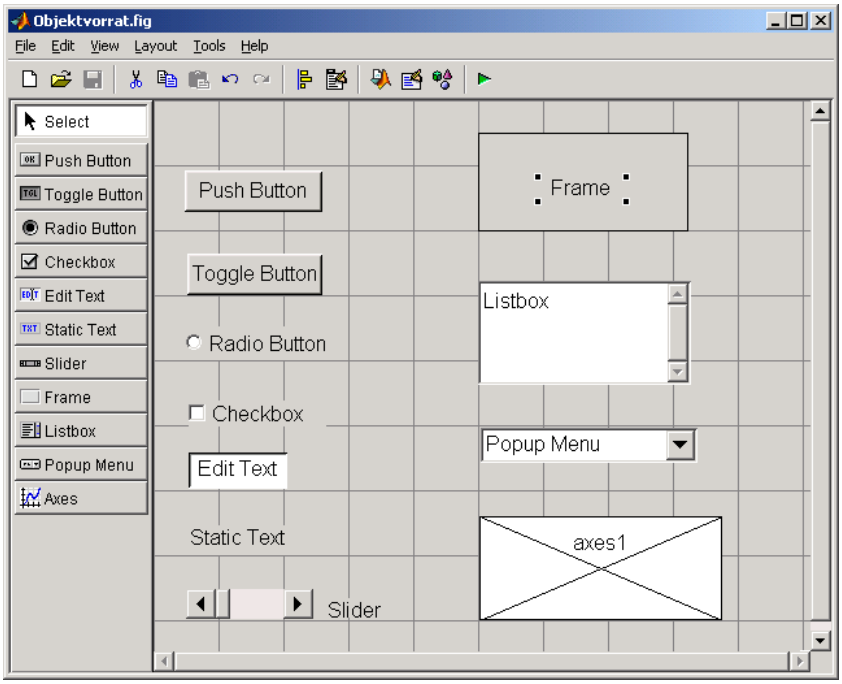

Abbildung 2: GUI-Editor

Die Eigenschaften der Komponenten bzw. Objekte des GUI können mit dem *Property Inspector* angezeigt und editiert werden. Dieser wird beispielsweise durch einen Doppelklick auf das entsprechende Objekt geöffnet (Abb. 3). Von der Vielzahl der Eigenschaften der Objekte seien hier nur einige besonders charakteristische angeführt:

Callback enthält den Funktionsaufruf, der bei jeder Aktivierung des Objekts durch Anklicken oder Ausführen einer Eingabe wirksam wird. Der Funktionsname wird automatisch generiert und vom Namen des Objekts (*Tag*) abgeleitet. Außerdem wird im Application-M-File automatisch eine entsprechende (leere) Funktion erzeugt.

String enthält einen Text zur Beschriftung einer Schaltfläche oder eines Auswahlfeldes.

Tag legt den Namen des Objekts fest. Er wird automatisch vergeben, sollte aber vom Programmierer durch eine problembezogene Bezeichnung ersetzt werden.

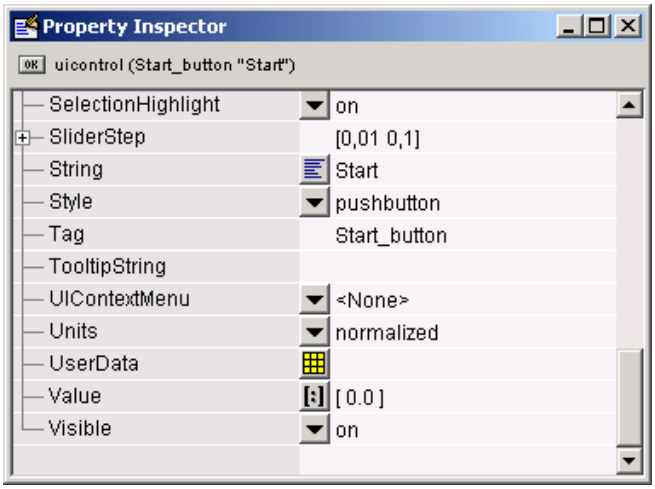

Abbildung 3: Property Inspector

TooltipString kann einen Text enthalten, der angezeigt wird, wenn sich der Mauszeiger für einige Zeit auf dem Objekt befindet.

Visible bestimmt, ob ein Objekt sichtbar oder unsichtbar ist. Das GUI-Layout kann somit zur Laufzeit durch Wechseln der Belegung von *Visible* verändert werden.

Beim ersten Speichern des Layouts werden zwei Dateien erzeugt: ein Fig-File mit der Beschreibung des entwickelten Layouts und ein Application-M-File, das die GUI-Figure aufruft und in dem auch schon bestimmte Funktionen der Objekte vordefiniert sind. Nachträgliche Änderungen des Layouts, z.B. von Objektnamen, bewirken i.a. auch automatisch eine entsprechende Änderung bzw. Anpassung des Application-M-Files. Allerdings funktioniert dieser Änderungsmechanismus nicht lückenlos. Namen in Kommentaren werden beispielsweise nicht angepasst.

4. Festlegen der GUI-Funktionalität

Bei jeder Aktivierung eines GUI-Objekts durch Anklicken oder Ausführen einer Eingabe wird ein so genannter Callback ausgeführt. Dieser ruft das Application-M-File auf und übergibt den Namen der betreffenden Callback-Funktion als Parameter. Beispielsweise erfolgt beim Betätigen des Start-Buttons im oben beschriebenen Beispiel (Abb.1) der Funktionsaufruf

messwertfilterung('Start_button_Callback',gcbo,[],guidata(gcbo))

Das File *messwertfilterung.m* ruft dann seinerseits die lokale Callback-Funktion *Start_button_Callback* auf, die im Beispiel wie folgt definiert ist:

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

global x1 t1
q1 = str2double(get(handles.qCurrentValue,'String'));
y(1) = x1(1);       % x1 Messwertvektor
for i = 2:101
    y(i)=x1(i)*(1-q1) + y(i-1)*q1;   % Exponentialfilter
end;
```

```
        % Create Plot Rohwerte
        axes(handles.Rohwerte)
        plot(t1, x1)
        set(handles.Rohwerte,'XMinorTick','on')
        grid on

        % Create Plot Filterwerte
        axes(handles.Gefilterte_Werte)
        plot(t1, y)
        set(handles.Gefilterte_Werte,'XMinorTick','on')
        grid on
```

Die ersten vier Zeilen der Funktion *Start_button_Callback* wurden vom GUIDE-Editor automatisch erzeugt, die restlichen wurden gemäß der zu lösenden Aufgabe unter Verwendung von MATLAB-Anweisungen bzw. -Funktionen und der im GUI-Layout festgelegten Objektnamen hinzugefügt.

Von besonderem Interesse sind auch die Programmabschnitte für die beiden Objekte, über die der Glättungsfaktor festgelegt werden kann, denn die Werte, die beiden zugeordnet sind, müssen immer übereinstimmen. Bei einer Änderung des Wertes des einen Objekts (z.B. *qValueSlider*) muss daher der Wert des anderen (z.B. *qCurrentValue*) nachgeführt werden. Im Application-M-File wird dieser Zusammenhang wie folgt hergestellt:

```
function qValueSlider_Callback(h, eventdata, handles)

  % Get the new value for qValueSlider from the slider
  q = get(h,'Value');

  % Set the value of the qCurrentValue to the new value set by slider
  set(handles.qCurrentValue,'String',q)


function qCurrentValue_CreateFcn(hObject, eventdata, handles)

% Hint: edit controls usually have a white background on Windows.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function qCurrentValue_Callback(hObject, eventdata, handles)

% Get the new value for qValueSlider
NewStrVal = get(hObject, 'String');
q = str2double(NewStrVal);
% Check that the entered value falls within the allowable range
if  isempty(q) | (q< 0) | (q>1),
   % Revert to last value, as indicated by qValueSlider
   OldVal = get(handles.qValueSlider,'Value');
   set(hObject, 'String',OldVal)
else, % Use new q value
   % Set the value of the qValueSlider to the new value
   set(handles.qValueSlider,'Value',q)
end
```

Aus obigem Programmstück wurden einige der automatisch eingefügten Kommentare aus Platzgründen und weil sie hier zur Erklärung nicht notwendig sind entfernt. Auf eine spezielle Erklärung der Programmzeilen wird verzichtet, weil der Text weitgehend selbsterklärend ist.

5. PEARL-Programm „Messwertfilterung“

Das Programm umfasst die drei Tasks *Start*, *Erfassen* und *Transfer*. Das Initialisieren und ordnungsgemäße Beenden des Programms übernimmt die Task *Start*. Die Task *Erfassen* ist für die zeitzyklische Erfassung des aktuellen Wertes der Messgröße, d.h. für die Bildung des Messwertvektors $x1$ und des Zeitvektors $t1$ zuständig. Alle Operationen, die mit Hilfe der MATLAB-Engine ablaufen, müssen in einer Task zusammengefasst sein. Die Task *Transfer* enthält daher neben den Anweisungen für die Übertragung der Vektoren auch die Funktionsaufrufe für das Öffnen und Schließen der Engine und das Starten des GUI „Messwertfilterung“. Weil diese Operationen im Gegensatz zu anderen nur einmal auszuführen sind, wird der Ablauf in der Task *Transfer* – wie aus dem dargestellten Programmfragment ersichtlich - mit Hilfe eines Flags gesteuert.

```
MODULE (messwertfilterung);
/* Beispiel fuer die Nutzung der MATLAB-Engine und eines MATLAB-GUI */
......
PROBLEM;
  DCL (x1, t1)(101) FLOAT;     ! x1 Messwerte, t1 Zeitmarken
  DCL i FIXED INIT(1);         ! Messwertindex
  DCL a FIXED;                 ! Flag für Steuerung der Task Transfer
  DCL (sema1, sema_trans) SEMA PRESET(1, 0);
.......
  /*** Funkionen der Datei engserv.c ***/
  SPC EngineOpen PROC GLOBAL;
  SPC EngineClose PROC GLOBAL;
  SPC EvalString PROC(REF INV CHAR()) GLOBAL;
  SPC VectorOut PROC(a() FLOAT IDENT, REF INV CHAR()) GLOBAL;

Start: TASK PRIO 5 MAIN;
  a=1;                ! Transfer-Flag Start Engine und Initialisierung
  ACTIVATE Transfer;  ! Starten der Engine und Initialisierung MATLAB
  SUSPEND;
  a=2;                ! Transfer-Flag Messwertuebertragung + Oeffnen GUI
  ALL Ta ACTIVATE Erfassen;
  AFTER 100*Ta ALL 101*Ta ACTIVATE Transfer;
......
END;
......
......
Transfer: TASK PRIO 50;
  CASE a
    ALT(1)  /* Aktivier. der Task Transfer am Anfang des Programms */
       EngineOpen;
       EvalString('global x1; global t1;');
       EvalString('path(path,''D:\PEARL90\MEngine_2004'');');
       CONTINUE Start;
    ALT(2)  /* Aktivier. von Transfer fuer 1. Messwertuebertragung */
       REQUEST sema_trans, sema1;
       VectorOut(x1, 'x1');
       VectorOut(t1, 't1');
       RELEASE sema1;
       EvalString('messwertfilterung;');  ! Oeffnen des GUI
       i = 1;                             ! Ruecksetzen Messwertindex
       a=3;
    ALT(3)  /* Aktivier. fuer 2. Messwertuebertragung und folgende */
```

```
       REQUEST sema_trans, sema1;
       VectorOut(x1, 'x1');
       VectorOut(t1, 't1');
       RELEASE sema1;
       i = 1;                          ! Ruecksetzen Messwertindex
    ALT(4)  /* Aktivier. von Transfer fuer Programmabschluss */
       EngineClose;
  FIN;
END;   ! Transfer
MODEND;
```

Die Parameter der C-Funktion *EvalString* entsprechen Kommandos, die beim Aufruf der Funktion auf der MATLAB-Ebene ausgeführt werden. Beim ersten Ausführen der Task *Transfer* nach dem Programmstart (a=1) werden

1. die MATLAB-Engine geöffnet,
2. die Variablen *x*1 und *t*1 als global deklariert, damit die Callback-Funktion *Start_button-Callback* darauf zugreifen kann und
3. der Pfad, in dem sich die GUI-Dateien befinden, in den MATLAB-Suchpfad aufgenommen.

Die C-Funktion *VectorOut* übergibt den als Parameter angegebenen Vektor unter dem durch den zweiten Parameter bezeichneten Namen an MATLAB.

6. Schlussbemerkung

Obige Beschreibung von MATLAB-GUI´s bezieht sich auf die MATLAB-Version 6.5. bzw. 6.5.1. Die seit einigen Wochen verfügbare Version 6.5.1 vereinfacht die Nutzung der MATLAB-Engine u.a. dadurch, dass die notwendigen Importbibliotheken *libeng.lib* und *libmx.lib* nicht mehr in der in [2] erläuterten, relativ aufwendigen Weise vom Programmierer erzeugt werden müssen, sondern im Verzeichnis \MATLAB\extern\lib\win32\ zur Verfügung gestellt werden; und zwar für verschiedene Plattformen bzw. Compiler. Außerdem verfügt MATLAB ab den genannten Versionen über neue, leistungsfähige Funktionen, mit denen Schnittstellen zu externen Programmen sich komfortabler und einfacher programmieren lassen [4].

7. Literatur

[1] Creating Graphical User Interfaces. Version 6. The Mathworks, Inc.
[2] Müller R.: Verwendung von MATLAB-Routinen in PEARL-Programmen. PEARL-News 1/2001.
[3] Angermann, A. u.a.: MATLAB – SIMULINK – Stateflow. Oldenbourg Verlag 2003.
[4} MATLAB – External Interfaces. Version 6. The Mathworks, Inc.

Prof. Dr.-Ing. R. Müller
HTWK Leipzig
Fachbereich Elektrotechnik und Informationstechnik
Wächterstraße 13, 04107 Leipzig
mueller@fbeit.htwk-leipzig.de