MS150 Modular Servo Workshop - Reference Manual

33-008-2M5

Feedback

- Computer Assisted Learning
- Electricity & Electronics
- Control & Instrumentation
- Process Control
- Mechatronics
- Robotics
- Telecommunications
- Electrical Power & Machines
- Test & Measurement

Technology Training for tomorrow's world

MS150 Modular Servo Workshop

Reference Manual

33-008-2M5

Feedback

Feedback Instruments Ltd, Park Road, Crowborough, E. Sussex, TN6 2QR, UK.
Telephone: +44 (0) 1892 653322, Fax: +44 (0) 1892 663719.
email: feedback@fdbk.co.uk World Wide Web Homepage: http://www.fbk.com

Manual: 33-008-2M5 Ed01 980930 *Printed in England by Fl Ltd, Crowborough*

Feedback Part No. 1160–330082M5

Notes

THE HEALTH AND SAFETY AT WORK ACT 1974

We are required under the Health and Safety at Work Act 1974, to make available to users of this equipment certain information regarding its safe use.

The equipment, when used in normal or prescribed applications within the parameters set for its mechanical and electrical performance, should not cause any danger or hazard to health or safety if normal engineering practices are observed and they are used in accordance with the instructions supplied.

If, in specific cases, circumstances exist in which a potential hazard may be brought about by careless or improper use, these will be pointed out and the necessary precautions emphasised.

While we provide the fullest possible user information relating to the proper use of this equipment, if there is any doubt whatsoever about any aspect, the user should contact the Product Safety Officer at Feedback Instruments Limited, Crowborough.

This equipment should not be used by inexperienced users unless they are under supervision.

We are required by European Directives to indicate on our equipment panels certain areas and warnings that require attention by the user. These have been indicated in the specified way by yellow labels with black printing, the meaning of any labels that may be fixed to the instrument are shown below:

CAUTION -
RISK OF
DANGER

Refer to accompanying documents

CAUTION -
RISK OF
ELECTRIC SHOCK

CAUTION -
ELECTROSTATIC
SENSITIVE DEVICE

PRODUCT IMPROVEMENTS

We maintain a policy of continuous product improvement by incorporating the latest developments and components into our equipment, even up to the time of dispatch.

All major changes are incorporated into up-dated editions of our manuals and this manual was believed to be correct at the time of printing. However, some product changes which do not affect the instructional capability of the equipment, may not be included until it is necessary to incorporate other significant changes.

COMPONENT REPLACEMENT

Where components are of a 'Safety Critical' nature, i.e. all components involved with the supply or carrying of voltages at supply potential or higher, these must be replaced with components of equal international safety approval in order to maintain full equipment safety.

In order to maintain compliance with international directives, all replacement components should be identical to those originally supplied.

Any component may be ordered direct from Feedback or its agents by quoting the following information:

1. Equipment type
2. Component value
3. Component reference
4. Equipment serial number

Components can often be replaced by alternatives available locally, however we cannot therefore guarantee continued performance either to published specification or compliance with international standards.

CE DECLARATION CONCERNING ELECTROMAGNETIC COMPATIBILITY

Should this equipment be used outside the classroom, laboratory study area or similar such place for which it is designed and sold then Feedback Instruments Ltd hereby states that conformity with the protection requirements of the European Community Electromagnetic Compatibility Directive (89/336/EEC) may be invalidated and could lead to prosecution.

This equipment, when operated in accordance with the supplied documentation, does not cause electromagnetic disturbance outside its immediate electromagnetic environment.

ACKNOWLEDGEMENTS

Feedback Instruments Ltd acknowledge all trademarks.

IBM, IBM - PC are registered trademarks of International Business Machines.

MICROSOFT, WINDOWS 95, WINDOWS 3.1 are registered trademarks of Microsoft Corporation.

MATLAB is a registered trademark of Mathworks Inc.

Table of contents

Notes

1 Introduction

The software of the Modular Servo Workshop model consists of:

- **The Real-Time Kernel (RTK),**
- **Communication functions,**
- **Modular Servo Toolbox,**
- **External Interface,**
- **Windows NT kernel-mode device driver (for Windows NT only).**

The Real-Time Kernel supervises a set of real-time tasks creating a feedback control structure. The RTK provides a mechanism of real-time measurements and control of the MS150 unit in the Windows 95 and Windows NT environments. It has the form of dynamic linked library (DLL). It is written in the closed form. No modifications are necessary. RTK contains all functions required for control and data collection in real-time. The RTK detects which operating system is running and changes its behaviour according to the operating system environment. Especially in the Windows NT environment the RTK kernel-mode device driver is started and enables access to the I/O address space.

Control algorithms are embedded in the Real-Time Kernel. Selection of control algorithms and tuning of their parameters is done by means of the **communication interface** from the MATLAB environment. The communication interface is also used for configuration of the Real-Time Kernel parameters (sampling period, excitation source etc.). Specialised communication functions are responsible for communication between RTK and MATLAB environment The functions are described in this manual.

Modular Servo Toolbox consist of a collection of M-functions (among them are S-functions as well), Simulink models and C-code DLL-files that extend the MATLAB environment in order to solve MS150 modelling/design/control problems. A cyclic memory buffer is created in order to enable process data flow between the real-time kernel and toolbox functions (the MATLAB environment).

The Modular Servo Workshop Toolbox, using MATLAB matrix functions and Simulink utilities, provides the user with functions specialised to real-time control of the modular DC Servo system.

It is the general assumption that the toolbox is an **open system (**in contrast to the RTK). This approach by its nature makes the basic functions of the toolbox available to the user.

However, demonstration M-files for typical MS150 control problems are available in the toolbox and can be invoked from the main control window.

The description of the demonstration files is given in the *Modular Servo Workshop Getting Started* manual – 33-008-1M5

External interface for the MS150 system offers the way to extend capabilities of the software. Although the RTK and MATLAB interface create a complete, self contained environment for real-time experiments its applicability is limited to a fixed number of embedded controllers. The interface forms the way in which user-designed controllers can be added to the RTK and implemented in real-time. The external interface is dedicated for the more experienced users.

The **kernel-mode device driver** is required in the Windows NT environment. The driver gives the RTK access to the I/O address space of the microprocessor.

The following sections contain the description of the communication functions. There are 32 communication functions listed in the alphabetic order. The functions are divided into four following categories according to the specific roles assigned to them.

- hardware: *GetBaseAddress, SetBaseAddress, ResetEncoder*
- data acquisition: *GetNoOfSamples, ResetTime, GetSampleTime, SetSampleTime, StartAcq, GetHistory*
- software: *LoadLibrary, UnloadLibrary*
- control: all 21 remaining functions.

The following format is used in this manual:

Purpose	Provides short description of the function
Synopsis	Shows the format of the function call
Description	Describes what the function does and any restrictions that apply
Arguments	Describes arguments of the function
See	Refers to other related functions
Examples	Provides examples of how the function can be used
Notes	Optionally remarks
Windows NT	Informs about limited applicability of the description. The description refers only to Windows NT

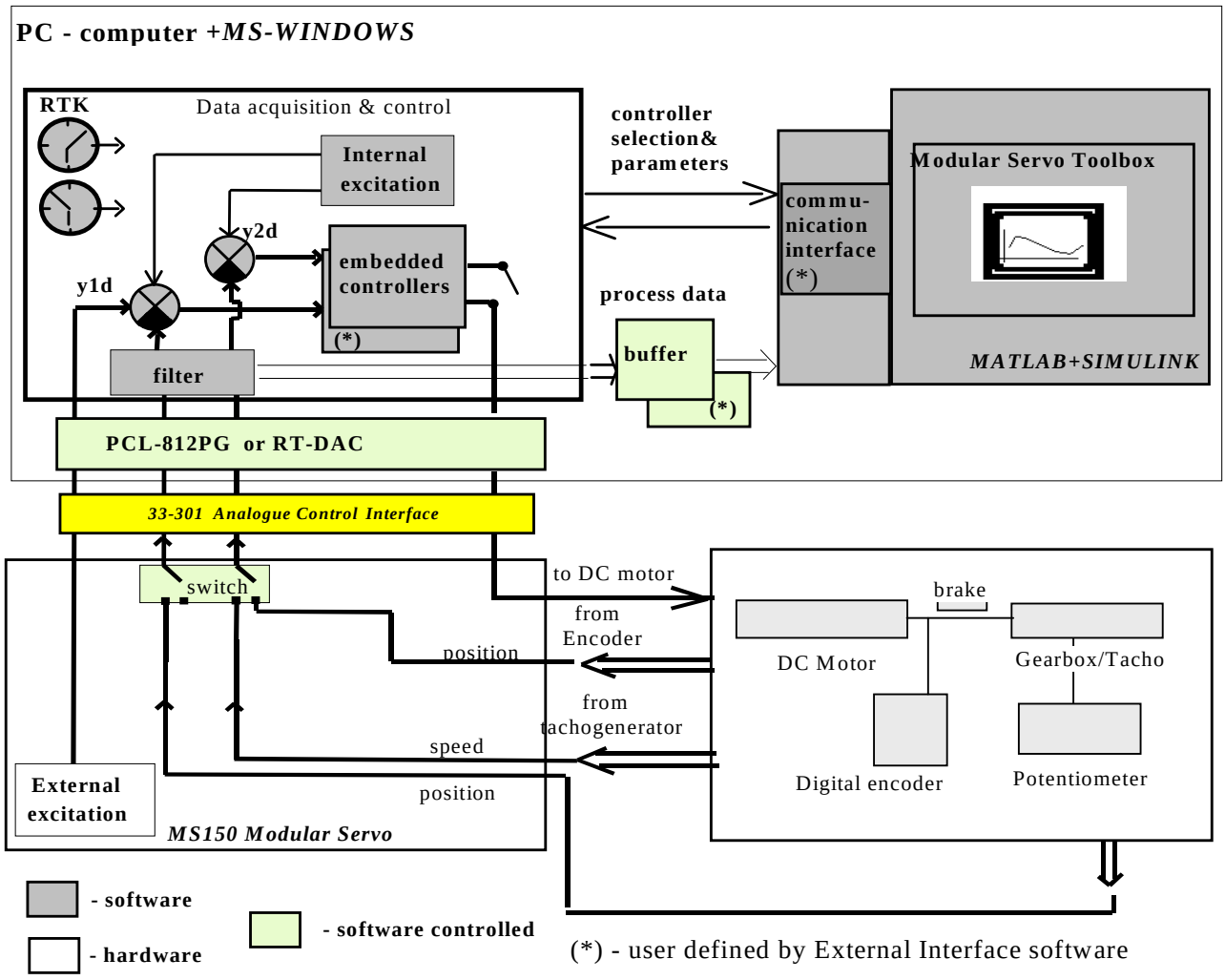


Figure 1-1: Digital control of the MS150 system (hardware and software configuration)

2 Description of the Toolbox Functions

All functions desired for communication between the Real-Time Kernel (RTK) and MATLAB environment have the following general form:

ReturnValue = es_call(FunctionName, [Argument])

where:

ReturnValue - value returned by the function,

es_call - name of the DLL library responsible for the communication,

FunctionName - name of the desired operation, string format. The *FunctionName* is case insensitive,

Argument - argument passed to the *es_call* function (optional).

GetAlgNo

Purpose: Get the number of the currently active control algorithm.

Synopsis: *AlgNo = es_call('GetAlgNo')*

Description: The function returns the number of the algorithm currently active in the RTK.

See: *SetAlgNo*

Example: The following function displays the description of the currently active control algorithm.

```
function dspdsr

  AlgNo = es_call( 'GetAlgNo' );
  if AlgNo == 0
    disp( 'Control set to zero' );
  elseif AlgNo == 1
    disp( 'Open-loop control' );
  elseif AlgNo == 2
    disp( 'PID controller' );
  elseif AlgNo == 3
    disp( 'LQ controller' );
  elseif AlgNo == 4
    disp( 'Time-optimal controller' );
  elseif AlgNo == 99
    disp( 'External controller' );
  else
    disp( '!!! Unknown controller !!!' );
  end
```

GetBaseAddress

Purpose: Get the base address of the input/output board from the RTK.

Synopsis: *BaseAddr = es_call('GetBaseAddress')*

Description: The function is called to obtain the base address of the RT-DAC or the PCL-812 interface board defined in the RTK.
If the base address of the I/O board is set to zero the RTK uses the built-in RTK simulator of the servo to generate data. This mode is useful to test communication between RTK and MATLAB without using any external hardware (see Example 2). When the simulator of the model is used the type of data which are collected in the built-in data acquisition buffer is defined by the last call to the *SetDataSource* function.

See: *SetBaseAddress, GetHistory*

Example1: *BaseAddr = es_call('GetBaseAddress')*

BaseAddr =

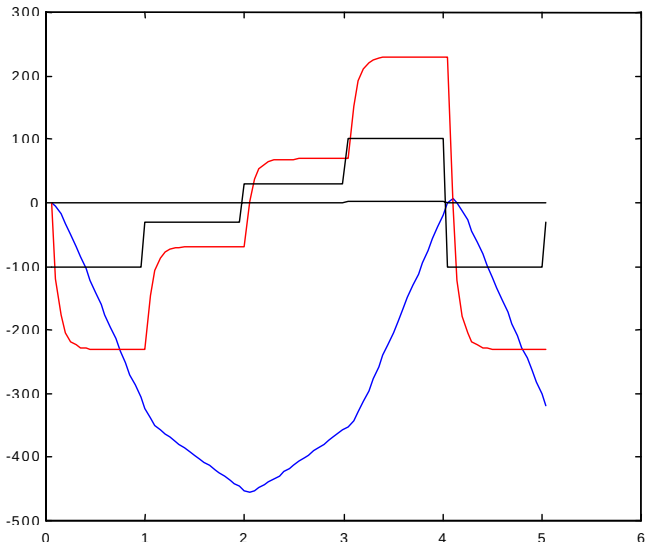
768

The current base address of the I/O board is set to 768 (or 300 hexadecimal)

Example 2: The base address equal to zero can be used to test the software. This operating mode is useful to test real-time software or to test MATLAB m-scripts and Simulink models. It does not require any interface hardware. The data are generated by the model of the real hardware which is built into the RTK.

The following commands display the example of data generated by the simulator.

```
ret = es_call( 'SetBaseAddress', 0 );
ret = es_call( 'SetPW', [ 1 1 1 1 1 -1 -0.3 0.3 1 zeros( 1, 11 ) ] );
ret = es_call( 'SetDataSource', [ 1 2 3 4 5 6 ] );
ret = es_call( 'SetAlgNo', 1 );
ret = es_call( 'StartAcq' );  pause( 5 )
hist = es_call( 'GetHistory' );
plot( hist( 1, : ), hist( [2 3 8], : ), hist( 1, : ), 100*hist( [8], : ) )
```



GetChan1Filt, GetChan2Filt, GetChan3Filt, GetChan4Filt, GetChan5Filt, GetChan6Filt

Purpose: Get the parameters of the digital filters.

Synopsis: *fi = es_call('GetChan1Filt')*

fi = es_call('GetChan2Filt')

fi = es_call('GetChan3Filt')

fi = es_call('GetChan4Filt')

fi = es_call('GetChan5Filt')

fi = es_call('GetChan6Filt')

1 **Description**: These functions get the parameters of the digital filters currently active in the RTK. The filter functions *GetChan1Filt, GetChan2Filt, GetChan3Filt, GetChan4Filt, GetChan5Filt* and *GetChan6Filt* are associated with the input channel 1, input channel 2, input channel 3, input channel 4, input channel 5 and input channel 6 respectively (see description of the *GetHistory* function).

The filter structure is described by the difference equation:

$$y(t) = a_0 x(t) + \sum_{i=1}^{8} a_i x(t - iT_0) + \sum_{i=1}^{8} b_i y(t - iT_0),$$

where:

$y(t)$ - filter output,

$x(t)$ - filter input,

T_0 - sampling period,

$a_0, \ldots, a_8, b_1, \ldots, b_8$ - parameters of the filter.

All filters implemented in RTK have parameter set to one and all others parameters set to zero, as default. It gives the transfer function for the filter equal to 1 (the output signal is equal to the input signal).

The return variable *fi* is a 2-by-9 matrix in the form:

$$\begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 \\ 0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 \end{bmatrix}.$$

See*:* *SetChan1Filt, SetChan2Filt, SetChan3Filt, SetChan4Filt, SetChan5Filt, SetChan6Filt*

GetDataSource

Purpose: Get information about data collected in the internal data acquisition buffer.

Synopsis: *src = es_call('GetDataSource')*

Description: The function returns the six-elements vector containing information about data collected in the internal data acquisition buffer. See the *SetDataSource* function for details..

See: *SetDataSource*

GetDivider

Purpose: Get the divider of the auxiliary clock.

Synopsis*:* *Div = es_call('GetDivider')*

Description: The function returns the divider of the basic RTK clock. This function allows to define a sampling period different then set by *SetSampleTime* function.

See*:* *SetDivider, GetSampleTime, SetSampleTime*

GetHistory

Purpose: Get content of the internal RTK buffer.

Synopsis: *Hist = es_call('GetHistory')*

Description: The function returns the *Hist* matrix containing the history of an experiment and sets the buffer to zero. The structure of the returned matrix *Hist* is given in Table 1.

The contents of the input channels is determined by the call to the *SetDataSource* function.

Table 1.

Row of the matrix	Description	Units
Hist(1,:)	regularly spaced time	seconds
Hist(2,:)	input channel 1	
Hist(3,:)	input channel 2	
Hist(4,:)	input channel 3	
Hist(5,:)	input channel 4	
Hist(6,:)	input channel 5	
Hist(7,:)	input channel 6	
Hist(8,:)	control for the DC drive	relative units in the range ±1

The maximum number of samples available in the buffer is 1024.

The internal data acquisition buffer becomes empty immediately after a *GetHistory* call. The call to the *GetNoOfSamples* function returns zero in this case.

See: *GetNoOfSamples, StartAcq*

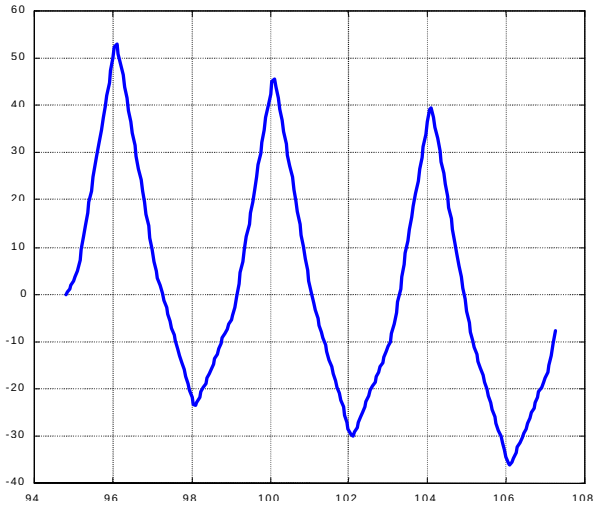
Example: Read 250 data points and display the data.

```
es_call( 'SetDataSource', [ 8 9 10 11 12 13 ] );
es_call( 'SetPW', [ 1 1 1 1 1 -0.5 -0.3 0.2 0.5 zeros(1, 11 ) ] );
es_call( 'SetAlgNo', 1 );
es_call( 'StartAcq' );                          % clear data buffer
while( es_call( 'GetNoOfSamples' ) < 250 ) ;    % wait for data
end;

Hist = es_call( 'GetHistory' );                 % get data and
                                                % clear buffer
es_call( 'StopPractical' );
```

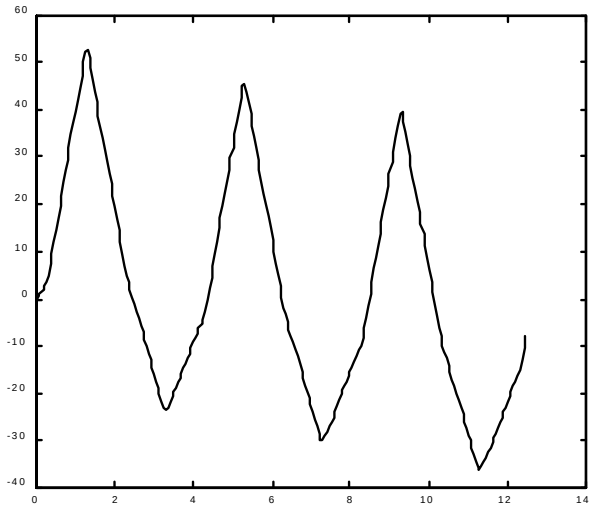
To plot angle of the DC shaft execute the following MATLAB command:

```
plot( Hist( 1, : ), Hist( 2, : ) ); grid
```



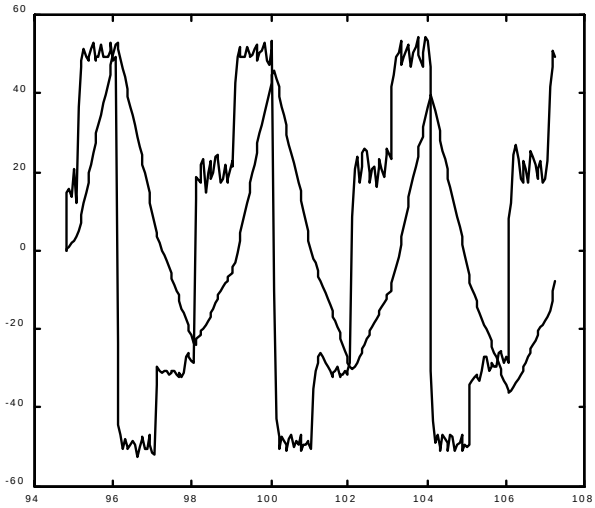
The figure above shows the angle vs. time diagram. Notice that the plot starts from approximately 94 seconds. This means that the experiment was started after 94 seconds from the moment when *es_call* library was loaded to the memory. If you want to plot the time from zero, simple modify the command:

plot(Hist(1, :)-Hist(1, 1),Hist(2, :))



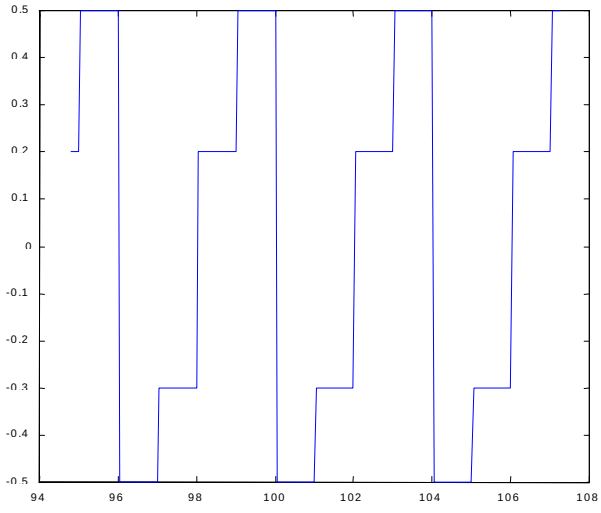
The following commands plot angle and angular velocity vs. time:

plot(Hist(1, :), Hist(2, :), Hist(1, :), Hist(3, :))

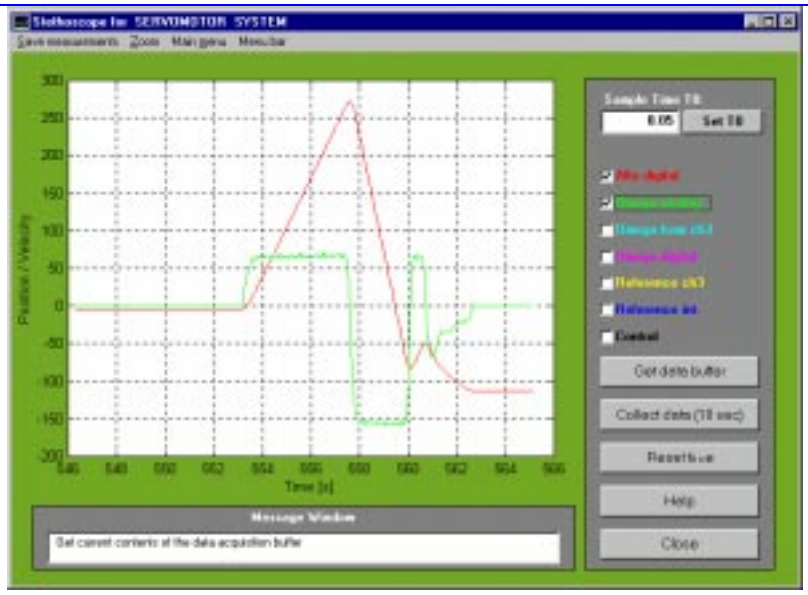


The following command plots the control value for the DC drive:

plot(Hist(1, :), Hist(8, :))



Note*:* There is a GUI interface available ready to plot data from the data acquisition buffer. Execute the command *es_stet* to display the following window:


Message Window
Close

GetModelP

Purpose: Return the parameters of the built-in mathematical model.

Synopsis: *Ret = es_call('GetModelP')*

Description: The function returns the vector containing four numbers. The elements of the vector corresponds to the mathematical model of the DC servo motor given by the following equations:

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx \end{cases}$$

where:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & \frac{-1}{T_S} \end{bmatrix},\ B = \begin{bmatrix} 0 \\ \frac{K_S}{T_S} \end{bmatrix},\ C = \begin{bmatrix} C_1 & 0 \\ 0 & C_2 \end{bmatrix},$$

x_1 is angle of the DC motor shaft,

x_2 is angular velocity,

u is input voltage,

y_1, y_2 are the outputs of the angle and velocity of the DC motor shaft.

See the Advanced Teaching Manual 33-008-4M5 for details.

The elements of the *Ret* vector are:

Ret(1) is equal to K_S,

Ret(2) is equal to T_S,

Ret(3) is equal to C_1,

Ret(4) is equal to C_2.

See: *SetModelP, SetInitCond, GetBaseAddress*

GetNoOfSamples

Purpose: Get the number of samples available in the buffer.

Synopsis: *Hist = es_call('GetNoOfSamples')*

Description: The function returns the number of samples currently available in the buffer. The maximum number of samples available in the buffer is equal to 1024.

See: *GetHistory*

Example *1:* Wait until the number of data in the buffer is greater than 120.

```
es_call( 'StartAcq' );                % clear buffer
                                      % length of buffer is set to zero
while( es_call( 'GetNoOfSamples' ) < 120 )
  ;
end;
```

Example *2:* The maximum data buffer length is 1024. The following sequence will never terminate:

```
while( es_call( 'GetNoOfSamples' ) < 1400 )
;
end;
```

GetP

Purpose: Get the parameters of the control algorithm.

Synopsis: *Par = es_call('GetP')*

Description: The function returns the vector containing the parameters of the control algorithm currently active in the RTK. The vector contains 20 elements. The set of the parameters is common for all algorithms, but only specified values are used by the currently active controller.

See: *GetPW, SetP*

Example: Increase the second parameter by 120%

```
Par = es_call( 'GetP' );
Par( 2 ) = 1.2 * Par( 2 );
es_call( 'SetP', Par );
```

GetPW

Purpose: Get the parameters of the internal excitation source.

Synopsis: *Par = es_call('GetPW')*

Description: The function returns the vector containing the parameters of the internal signal generator currently active in the RTK. The vector contains 20 elements. The set of parameters is common for all shapes of signal waves, but only some values are used by the currently active excitation source. The internal excitation source may be used as:

- a source of control value for the DC motor in the open-loop mode,
- a source of reference angle in the closed-loop mode.

See: *SetPW*

Example: Set and read the parameters of the internal excitation source.

```
Par = zeros( 1, 20 );
Par( 1 : 5 ) = [ 1 -1.2 4.66 0 7 ];
es_call( 'SetPW', Par);
Par = es_call( 'GetPW' )
Par =
    1 -1.2 4.66 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

GetSampleTime

Purpose: Get the basic sampling time.

Synopsis: *SmplT = es_call('GetSampleTime')*

Description: The function returns the period of the basic RTK clock. The period is given in seconds.

See: *SetSampleTime*

GetV2Velocity

Purpose: Get coefficient which is used to calculate velocity of the DC motor shaft based on the voltage generated by the tachogenerator.

Synopsis: *v2 = es_call('GetV2Velocity')*

Description: The function returns a scaling coefficient used to calculate velocity of the motor shaft based on the voltage generated by the tachogenerator. The output voltage from the tachogenerator is multiplied by this coefficient and divided by 5.0 to calculate velocity of the shaft expressed in [deg/sec].

See: *SetV2Velocity*

LoadLibrary

Purpose: Load RTK to memory.

Synopsis: *Count = es_call('LoadLibrary')*

Description: The function loads the RTK to the memory. If you wish to finish the RTK, you should use the *UnloadLibrary* function.

First call to any *hl_call* function loads the RTK into memory automatically. All functions of the RTK are included in the *hl_call.dll* module.

Note: The following messages can appear during execution of the *LoadLibrary* function:

Can not open ES_PAR.INI file. Base address is set to zero - the *es_par.ini* file is not present in the directory which contains the *es_call.dll* file. The es_*par.ini* file should contain the following line which defines base address of the I/O board:

BaseAddress= 768

If the *es_par.ini* file is missing the base address of the board is set to zero. It causes the RTK to generate dummy data (see the *GetBaseAddress* function).

If the *es_par.ini* file contains the line:

RT-DAC

the RTK uses the RT-DAC I/O board instead of the PCL-812PG board.

All lines in the *es_par.ini* file which begin with the '*%'* character are comments,

Can not find BaseAddress parameter in the ES_PAR.INI file - the *es_par.ini* file is corrupted. See message above for details,

Windows NT

Couldn't access RTKIO device, or

Can not start IO access - these two messages can appear in the Windows NT operating system only. They are caused by absence of the *RTKIO* kernel mode device driver properly installed and running in the operating system. See description of the installation procedure to solve this problem.

See*:* *UnloadLibrary*

ResetEncoder

Purpose: Set the incremental encoder to the initial position.

Synopsis*:* *es_call('ResetEncoder')*

Description: The function defines the initial position of the DC servo. The function is called while the system is in a steady-state. The *ResetEncoder* function should be executed immediately after loading RTK to the memory.

See*:* *LoadLibrary*

ResetTime

Purpose: Set the experiment time to zero.

Synopsis: *es_call('ResetTime')*

Description: The function sets the time counter of the RTK to zero. Time is calculated from the first call of the *es_call* function. After a call of the *ResetTime* function the time vector returned by *GetHistory* function starts from zero.

See: *GetHistory*

SetAlgNo

Purpose: Select a control algorithm and start the experiment.

Synopsis*:* *es_call('SetAlgNo', AlgNo)*

Description: The function is called to select a control algorithm embedded in the RTK. The list of available algorithms is given in Table 2. The function starts the experiment.

The algorithm is either supplied with appropriate parameters, or default parameters are used. The *SetAlgNo* function must be preceded by a call of the *SetP* or *SetPW* functions. Use the *SetPW* function before *es_call('SetAlgNo', 1)*, and *SetP* in other cases.

Table 2.

Algorithm no	Description
0	Stop experiment - set control to zero
1	Open loop excitation. Uses internal generator as control signal sources
2	PID controller.
3	state feedback controller
4	time optimal
99	External controller

The **PID** controller uses the data available in the first input channel (see *GetHistory* and *SetDataSource* functions) as the variable which is stabilised and the data available in the third channel of the data acquisition buffer as the reference value. **Because the first data**

acquisition channel may contain position or velocity data the PID controller may be used to stabilise the position or the velocity.

The **state feedback** controller uses the data available in the first input channel of the data acquisition buffer as position of the motor shaft, the data available in the second channel of the data acquisition buffer as the velocity of the shaft and the data available in the third channel as the reference value.

The **time optimal** controller interprets the data in the internal buffer in the same way as the state feedback controller does.

See: *GetAlgNo, GetP, GetPW, SetP, SetPW*

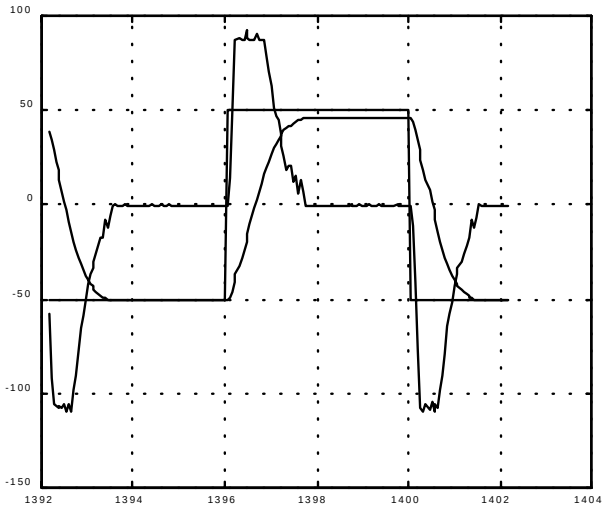
Example 1: Activate the internal excitation source and generate a triangle control signal. Start an open-loop experiment.

```
es_call( 'SetAlgNo', 0 );                  % stop previous
                                           % experiment
es_call( 'SetPW', [ 2 5 1 -0.7 0.5] );     % set triangle excitation
                                           % for horizontal
                                           % position
es_call( 'SetAlgNo', 1 );                  % start experiment
es_call( 'GetAlgNo' )
ans =
     1
```

Example 2: Set the PID controller parameters and start a close-loop experiment.

```
es_call( 'SetDataSource', [ 8 9 13 1 1 1 ] );
es_call( 'SetPW', [ 1 2 2 2 2 -50 -50 50 50 zeros( 1, 11 ) ] );
es_call( 'SetAlgNo', 0 );            % stop experiment
p = [ 0.025 3.46 0.013 1 1 zeros( 1, 15 )];
p = es_call( 'SetP', p );

es_call( 'SetAlgNo', 2 );            % begin PID experiment
es_call( 'StartAcq' );               % clear data buffer
pause( 10 );
Hist = es_call( 'GetHistory' );      % get data and plot
plot( Hist(1,:), Hist( 2:4, : ) ); grid
```



SetBaseAddress

Purpose: Define in the RTK the base address of the RT-DAC or the PCL-812 board.

Synopsis: *es_call('SetBaseAddress', Address)*

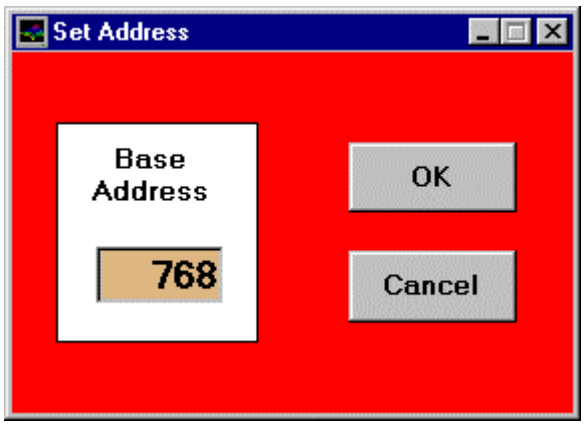
Description: The function is called to define the base address of the PCL-812 or the RT-DAC board. Notice that the base address is fixed in the configuration file *es_par.ini* and is read each time RTK is loaded to memory. If the base address is set to zero the RTK generates „dummy" data which are created by the software simulator of the real model.

See: *GetBaseAddress*

Example: Set the address of the I/O board to 768 (220 in hexadecimal code).

```
es_call( 'SetBaseAddress', 768 );
es_call( 'GetBaseAddress' )
ans =
      768
```

Note*:* There is a GUI interface available, which may be used to set new value of the base address. To call it execute the *es_sad* command. The result is shown below.



SetChan1Filt, SetChan2Filt, SetChan3Filt, SetChan4Filt, SetChan5Filt, SetChan6Filt

Purpose: Set the parameters of the digital filters.

Synopsis: *ret = es_call('SetChan1Filt', fi)*

ret = es_call('SetChan2Filt', fi)

ret = es_call(SetChan3Filt', fi)

ret = es_call('SetChan4Filt', fi)

ret = es_call('SetChan5Filt', fi)

ret = es_call('SetChan6Filt', fi)

Description: These functions set the parameters of the digital filters currently active in RTK. The filter are associated with appropriate input channels. The filter structure is described by the difference equation:

$$y(t) = a_0 x(t) + \sum_{i=1}^{8} a_i x(t - iT_0) + \sum_{i=1}^{8} b_i y(t - iT_0) ,$$

where:

$y(t)$ - filter output,

$x(t)$ - filter input,

T_0 - sampling period,

$a_0, \ldots, a_8, b_1, \ldots, b_8$ - parameters of the filter.

All filters implemented in RTK have a_0 parameter set to one and all others parameters set to zero as default. It gives the transfer function for the filter equal to 1 (the output signal is equal to the input signal).

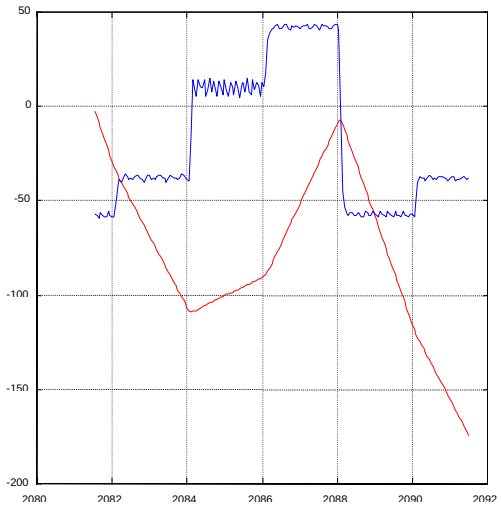
If you need other filter transfer function the *fi* variable must be set. For this purpose the argument *fi* must be defined as 2-by-9 matrix in the form:

$$\begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 \\ 0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 \end{bmatrix}.$$

See: *GetChan1Filt, GetChan2Filt, GetChan3Filt, GetChan4Filt, GetChan5Filt, GetChan6Filt*

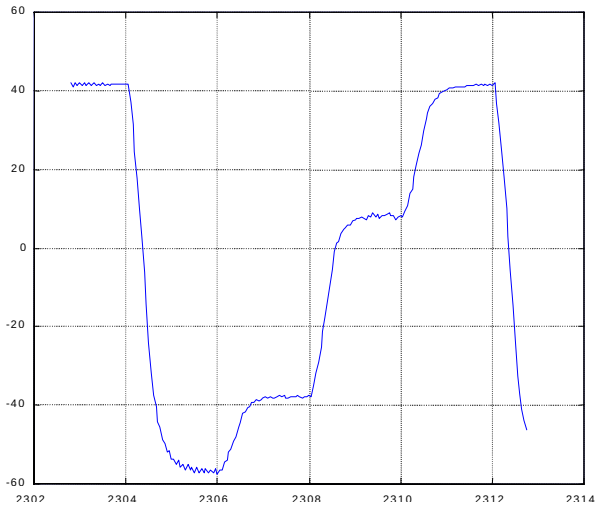
Example: Execute the following commands to collect and plot angle and velocity of the motor shaft calculated from the incremental encoder channel:

```
es_call( 'SetDataSource', [ 8 11 1 1 1 1 ] );
es_call( 'ResetEncoder' );
es_call( 'SetPW', [ 1 2 2 2 2 -0.5 -0.3 0.2 0.5 zeros(1, 11 ) ] );
es_call( 'SetAlgNo', 1 );
es_call( 'StartAcq' );                    % clear data buffer
pause( 10 )
Hist = es_call( 'GetHistory' );           % get data and
plot( Hist( 1, : ), Hist( 2 : 3 , : ) ); grid
```



Notice the difference in the velocity data after the following commands:

```
fi = [ 0.125 0.125 0.125 0.125 0.125 0.125 0.125 0.125 ; ...
       0 0 0 0 0 0 0 0 ];
es_call( 'SetChan2Filt', fi );
es_call( 'StartAcq' );                  % clear data buffer
pause( 10 )
Hist = es_call( 'GetHistory' );         % get data and
plot( Hist( 1, : ), Hist( 3 , : ) ); grid
```



SetDataSource

Purpose: Define the source of data collected in the internal data acquisition buffer.

Synopsis: *src = es_call('SetDataSource', data_source)*

Description: The function accepts the six-elements vector defining the source of data collected in the internal data acquisition buffer. The values in the *data_source* input argument have the meaning described in Table 3.

Table 3.

Value	Description
1	voltage of the signal connected to the analogue input channel 10 of the I/O board expressed in [V]
2	voltage of the signal connected to the analogue input channel 11 of the I/O board expressed in [V]
3	voltage of the signal connected to the analogue input channel 12 of the I/O board expressed in [V]
4	voltage of the signal connected to the analogue input channel 13 of the I/O board expressed in [V]
5	voltage of the signal connected to the analogue input channel 14 of the I/O board expressed in [V]

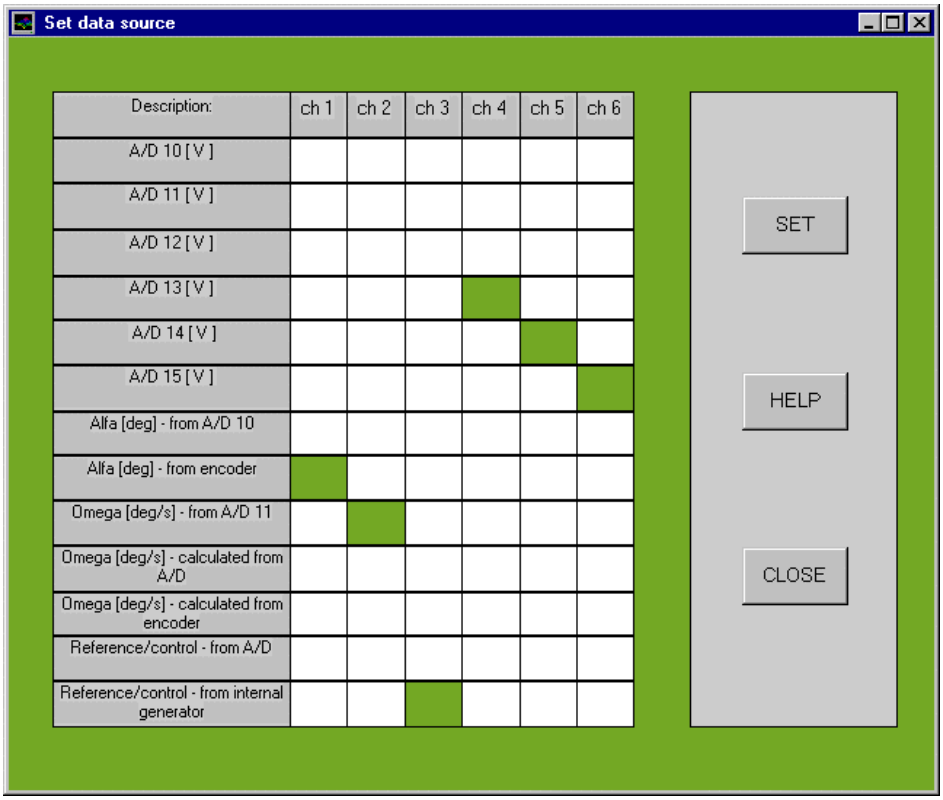
6	voltage of the signal connected to the analogue input channel 15 of the I/O board expressed in [V]
7	angle expressed in [deg] - signal connected to the analogue input channel 11 of the I/O board
8	angle expressed in [deg] maesured from incremental encoder sensor
9	angular velocity from tachogenerator connected to the analogue input channel 11 of the I/O board expressed in [deg/sec]
10	angular velocity calculated from position signal connected to the analogue input channel 3 (channel A/D 11 of the I/O board) of the I/O board expressed in [deg/sec]
11	angular velocity calculated from position signal obtained from incremental encoder expressed in [deg/sec]
12	reference value from the signal connected to the analogue input channel 12 of the I/O board expressed in [deg]
13	internal excitation generator

Example: Collect data from analogue input channels 10, 11, 12, 13, 14 of the I/O board and velocity calculated based on the signal from incremental encoder.

ret = es_call('SetDataSource', [1 2 3 4 5 11]);

See: *GetDataSource*

Note*:* The user can use the *es_src* MATLAB function to set data source using MATLAB GUI interface. The appropriate window is shown below:



SetDivider

Purpose: Set the clock divider.

Synopsis*:* *es_call('SetDivider', Div)*

Description: The function sets an auxiliary clock of RTK. This function allows the definition of a control period different from the value defined by the *SetSampleTime* function. This clock can be used by control algorithms.

The *Div* argument must be a positive integer number.

See*:* *GetDivider*

Example 1: Set the control period 3 times longer than the sampling period.

es_call('SetDivider', 3);

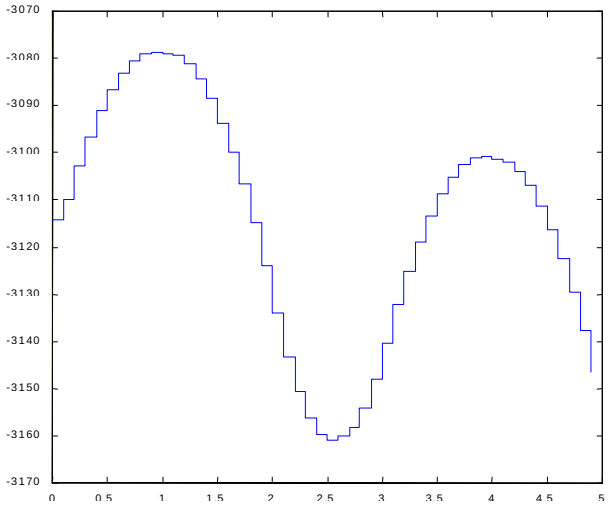
Example 2: Set the control period equal to the sampling period.

es_call('SetDivider', 1);

Example 3: The following sequence of commands:

```
% select open-loop control
es_call( 'SetDivider', 1 );
% set triangle open-loop excitation
es_call( 'SetPW', [2 1 2 -1 1 zeros( 1, 15 ) ] );
es_call( 'SetDataSource', [8 1 1 1 1 1 ] );
es_call( 'SetSampleTime', 0.1 );
es_call( 'SetAlgNo',1 );
es_call( 'StartAcq' );
pause( 5 )
h = es_call( 'GetHistory' );
stairs( h( 1, : )-h( 1, 1 ),h( 2, : ) )
```

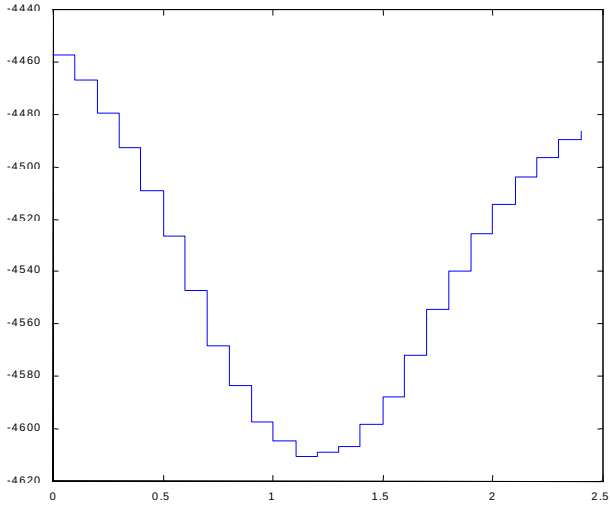
creates the following diagram:



After the commands:

```
es_call( 'SetDivider', 2 )
es_call( 'StartAcq' );
pause( 5 )
h = es_call( 'GetHistory' );
stairs( h( 1, : )-h( 1, 1 ),h( 2, : ) );
```

the picture changes (see figure below). Notice that the control value changes three times slower.



SetInitCond

Purpose: Set the initial conditions for the mathematical model.

Synopsis: *Ret = es_call('SetInitCond', par)*

Description: The function sets the initial conditions for the mathematical model built into the RTK. The new initial conditions are applied starting from the next sample time step. The *par* vector contains two elements:

par(1) is initial value of the angle, and

par(2) is initial value for the velocity.

See: *GetModelP, SetModelP*

SetModelP

Purpose: Set the parameters of the built-in mathematical model.

Synopsis: *Ret = es_call('SetModelP', par)*

Description: The function sets the parameters of the mathematical model. The model is used by the internal system simulator and is given in the form of the following linear equations:

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx \end{cases}$$

where:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & \dfrac{-1}{T_S} \end{bmatrix},\ B = \begin{bmatrix} 0 \\ \dfrac{K_S}{T_S} \end{bmatrix},\ C = \begin{bmatrix} C_1 & 0 \\ 0 & C_2 \end{bmatrix},$$

x_1 is angle of the DC motor shaft,

x_2 is angular velocity,

u is input voltage,

y_1, y_2 are the outputs of the angle and velocity of the DC motor shaft.

The elements of the *par* vector are:

par(1) is equal to K_S,

par(2) is equal to T_S,

par(3) is equal to C_1,

par(4) is equal to C_2.

See: *GetModelP, SetInitCond*

SetP

Purpose: Set the parameters of the controller.

Synopsis: *es_call('SetP', Par)*

Description: The function is called to set parameters of the controller. Next, the number of the algorithm should be selected (*SetAlgNo*). The selected algorithm will be activated with the pre-set parameters. The function *SetP* can be used for all control algorithms. The controller parameters have the following meaning:

- for algorithm number 0 (set control value to zero):

  the parameters are not required,

- for algorithm number 1 (open-loop control):

  see function *SetPW* detailed description,

- for algorithm number 2 (PID controller) the control value is calculated due to the following algorithm:

$$\varepsilon_n = Ch3_n - Ch1_n,$$

$$I_n = \sum_{k=1}^{n} T_0 * \varepsilon_k ,$$

$$if\ (I_n > Isat)\ then\ I_n = Isat ,$$

$$if\ (I_n < -Isat)\ then\ I_n = -Isat ,$$

$$U_n = Kp * \varepsilon_n + Ki * I_n + Kd * \frac{\varepsilon_n - \varepsilon_{n-1}}{T_0},$$

$$if\ (U_n > U\ \max)\ then\ U_n = U\ \max ,$$

$$if\ (U_n < -U\ \max)\ then\ U_n = -U\ \max ,$$

where:

$Ch1_n$, $Ch3_n$ are current values from the first and third input channel in the data acquisition buffer,

ε_n denotes control error in the current sampling period,

Kp, Ki, Kd, $Isat$ and $U\max$ are parameters of the PID controller,

T_0 is sampling period.

The elements of the *Par* vector are equal respectively:

Par(1) is equal to the *Kp* parameter,
Par(2) is equal to the *Ki* parameter,
Par(3) is equal to the *Kd* parameter,
Par(4) is equal to the *Imax* parameter,
Par(5) is equal to the *Umax* parameter,

- for algorithm number 3 (state feedback controller) the control value is calculated due to the following algorithm:

$$\varepsilon_n = Ch3_n - Ch1_n,$$

$$U_n = K1 * \varepsilon_n - K2 * Ch2_n,$$

$$if\ (U_n > U\max)\ then\ U_n = U\max,$$

$$if\ (U_n < -U\max)\ then\ U_n = -U\max,$$

where:

$Ch1_n$, $Ch2_n$ and $Ch3_n$ are current values from the first, second and third input channels,

ε_n denotes control error in the current sampling period,

$K1$, $K2$ and $U\max$ are parameters.

The elements of the *Par* vector are equal respectively:

Par(1) is equal to the *K1* parameter,
Par(2) is equal to the *K2* parameter,
Par(3) is equal to the *Umax* parameter,

- for algorithm number 4 (time optimal controller) the control value is calculated using the algorithm described in the Advanced Teaching Manual – 33-008-4M5

See: *GetP, GetPW*

Example: Start PI controller for the DC drive.

```
es_call( 'SetDataSource', [ 8 9 13 1 1 1 ] );
es_call( 'SetPW', [ 1 2 2 2 2 -50 -50 50 50 zeros( 1, 11 ) ] );
es_call( 'SetAlgNo', 0 );          % stop experiment
p = [   0.025 ...       % Kp
        3.46 ...            % Ki
        0.0 ...         % Kd
        1 ...           % Isat
        1...            % Umax
         zeros( 1, 15 )];
p = es_call( 'SetP', p );
es_call( 'SetAlgNo', 2 );          % begin PID experiment
```

SetPW

Purpose: Set the parameters of the internal excitation.

Synopsis: *es_call('SetPW', Par)*

Description: The function sets the parameters of the internal signal source. The internal signal source is used as:

- a source of control value for the DC drive in the case of open-loop control mode (algorithm number 1),
- a source of reference value for the position in the case of all closed-loop control algorithms.

The following periodical functions can be selected: constant, square, triangle, sinusoidal or random.

The elements of the *Par* argument have the following meaning:

Par(1) = 0: constant value. In this case (Figure 2-1):
Par(2) is the level of the signal,

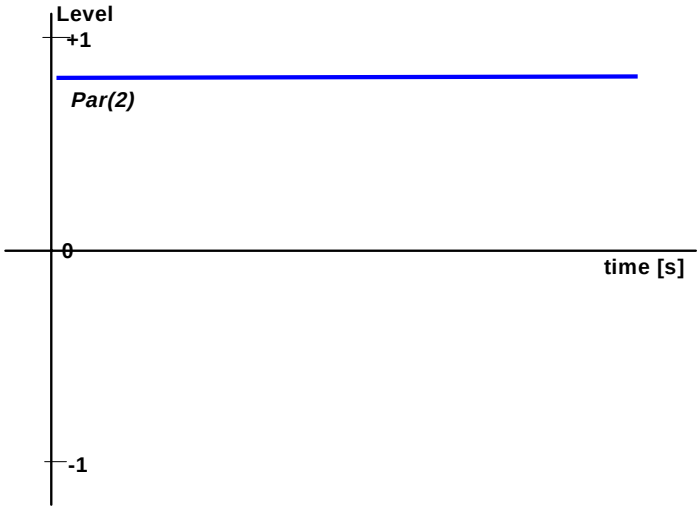


Figure 2-1: Constant excitation.

Par(1) = 1: square wave. In this case (Figure 2-2):
Par(2), Par(3), Par(4), Par(5) - time periods,
Par(6), Par(7), Par(7), Par(8) - appropriate levels of the signal,

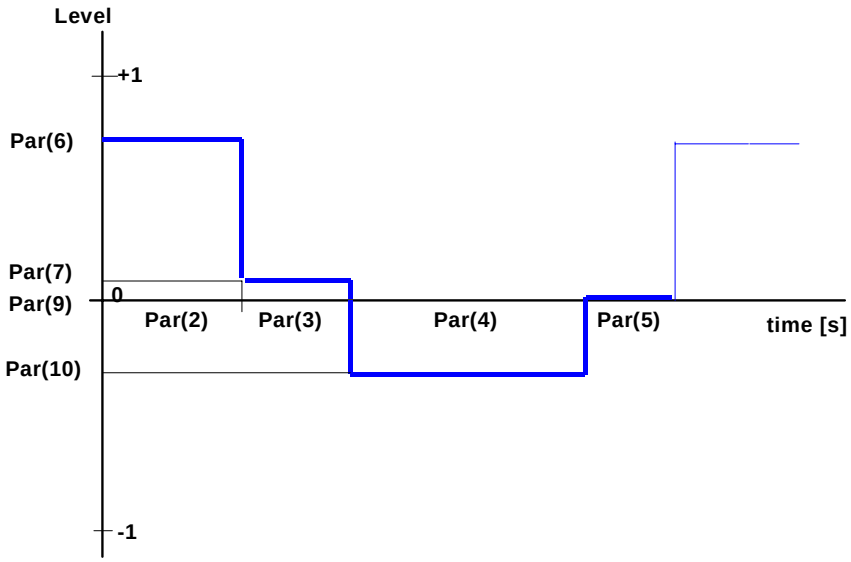


Figure 2-2: Square excitation.

Par(1) = 2: triangle wave. In this case (Figure 2-3):
Par(2), Par(3) - time periods,
Par(4), Par(5) - appropriate levels,

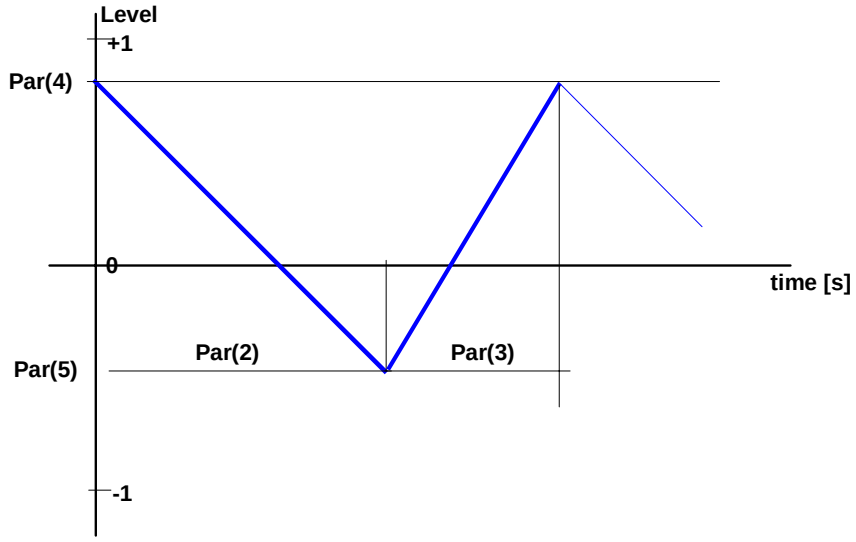


Figure 2-3: Triangle excitation.

Par(1) = 3: sinusoidal wave. In this case (Figure 2-4):
Par(2) - time period,
Par(3), Par(4) - maximum and minimum values of the signal,

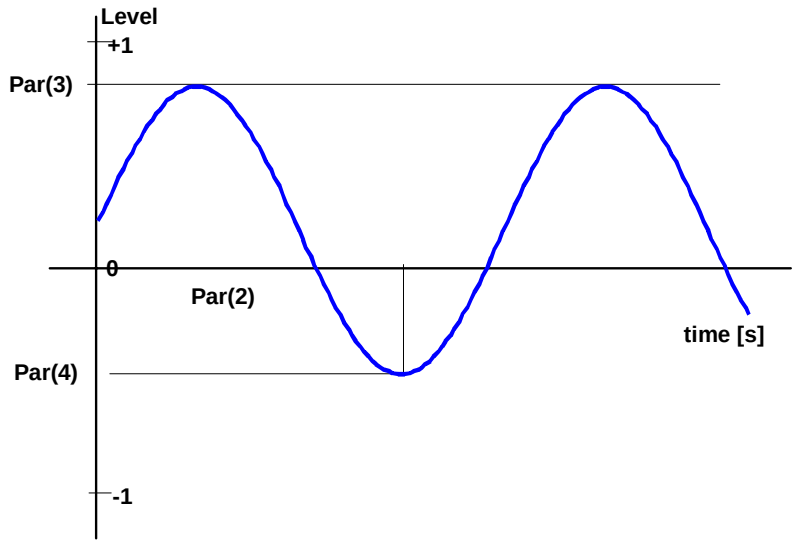


Figure 2-4: Sinusoidal excitation.

Par(1) = 4: random wave. In this case (Figure 2-5):

Par(2) - time period when the output of the random generator is kept constant,

Par(3), Par(4) - maximum and minimum values of the random signal.

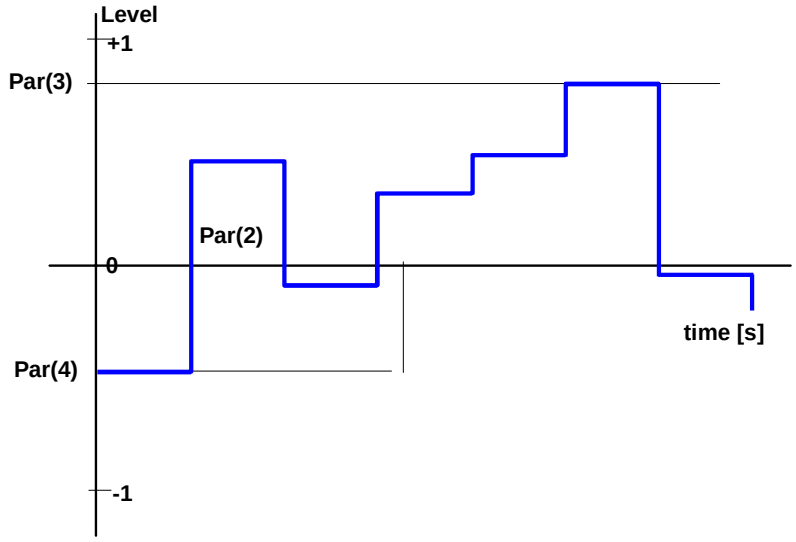


Figure 2-5: Random excitation.

See: *GetPW, SetAlgNo*

Example: Set the internal excitation source to generate a sinusoidal wave: period equals to 4 seconds, minimum value is -0.3, maximum value is 0.9.

```
Par = es_call( 'GetPW' );
Par( 1 : 4 ) = [ 3  4  -0.3  0.9 ];
es_call( 'SetPW', Par);
es_call( 'GetPW' )
ans =
      3  4  -0.3  0.9  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
es_call( 'SetAlgNo', 1 );          % Open-loop control
```

SetSampleTime

Purpose: Set the basic clock.

Synopsis: *es_call('SetSampleTime', Period)*

Description: The function sets the basic clock of the RTK. The sampling period of A/D converters is set by this function. The controller output rate (D/A) can be equal to or greater than the basic clock frequency. The *Period* parameter must be in the range from 0.001s to 32.767s. The lower bound depends on the hardware configuration. The resolution is 0.001s.

See: *SetDivider, GetSampleTime*

Example: To set and to read the sampling period use the following statements:

```
es_call( 'SetSampleTime', 0.025 );
es_call( 'GetSampleTime' )
ans =
        0.025
es_call( 'SetSampleTime', 0.0 );% SampleTime<0.001 results
es_call( 'GetSampleTime' )      % in 0.001
ans =
        0.001
```

SetV2Velocity

Purpose: Set coefficient used to calculate velocity of the motor shaft based on the voltage generated by the tachogenerator.

Synopsis: *ret = es_call('SetV2Velocity', coeff)*

Description: The function sets the coefficient used to calculate velocity of the motor shaft based on the voltage generated by the tachogenerator. The output voltage from the tachogenerator is multiplied by this coefficient and divided by 5.0 to calculate velocity of the disk, expressed in [deg/sec].

The call to this function sets the new value of the coefficient only for the current session of the RTK. When you remove the RTK library from the memory and load it again you have to call this function again.

To make the definition of this coefficient permanent the following line have to be added to the *es_par.ini* file:

V2Velocity= new_value_of_coeffictient

The typical value of the coefficient is equal approximately to 1040.

See: *GetV2Velocity*

StartAcq

Purpose: Clear the content of the data acquisition buffer.

Synopsis*:* *es_call('StartAcq')*

Description: The function erases the content of the buffer. The action is similar to that of the *GetHistory* function, but does not return the content of the buffer. Notice, that the *GetNoOfSamples* function returns zero when invoked immediately after the *StartAcq* call**.**

See*:* *GetNoOfSamples*

StopPractical

Purpose: Stop the practical.

Synopsis*:* *es_call('StopPractical')*

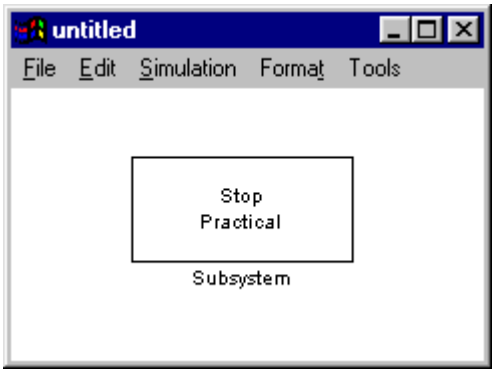
Description: The function is called to stop the practical. It suspends the currently active RTK control algorithm and sets control values for DC motor to zero.

The action is similar to

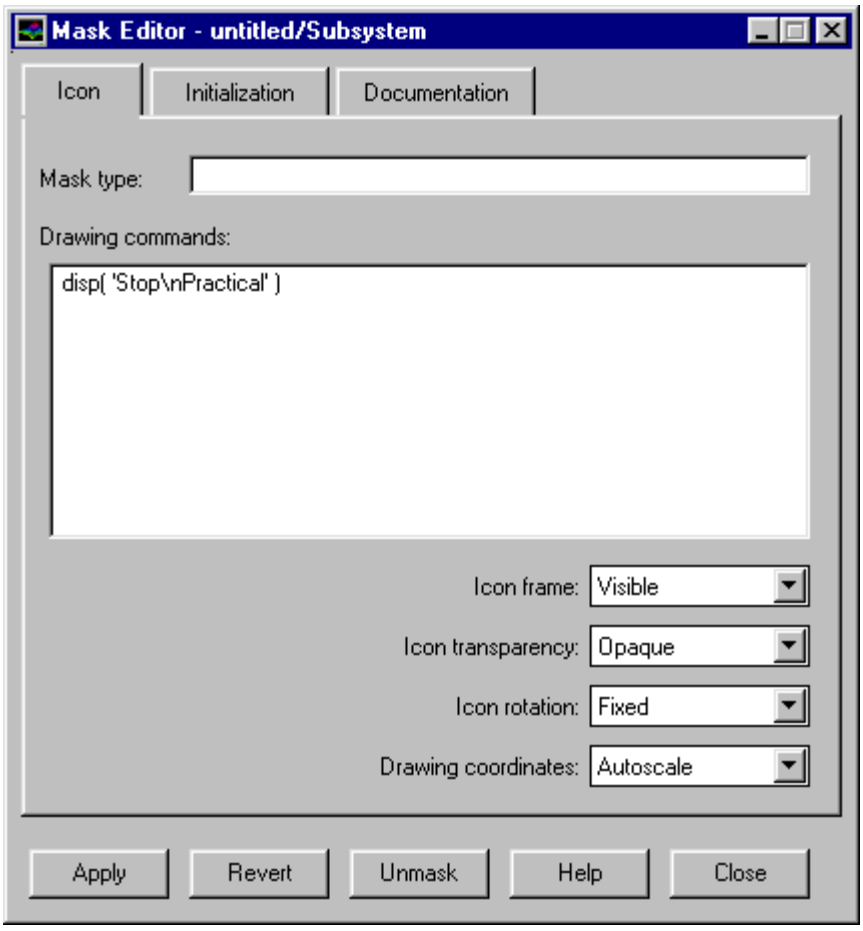
es_call('SetAlgNo', 0).

See*:* *SetAlgNo*

Example*:* You can create the Simulink model containing single block which can be used to stop experiments. Double click over this block will cause execution of the *es_call('StopPractical')* function. The Simulink model is shown below.



The *Stop Practical* block is masked. The mask of the *Subsystem* block sets the inscription of the function inside the block. It is shown below.



To join the *StopPractical* block with the *StopPractical* action the following MATLAB command must be executed:

set_param('untitled/Subsystem', 'OpenFcn' , ... 'es_call("StopPractical");')

This command sets the *OpenFcn* property of the *Subsystem* block from the *untitled* Simulink model to the *es_call('StopPractical')* statement. It causes execution of the *es_call('StopPractical')* command when the user double clicks over the *Subsystem* block.

UnloadLibrary

Purpose: Remove the RTK library from memory.

Synopsis: *Count* **=** *es_call('UnloadLibrary')*

Description: The function removes immediately the RTK library from the memory. The termination of MATLAB removes the RTK from the memory.

See: *LoadLibrary*

3 Information Flow Between Simulink Models and the Real-Time Kernel

This section describes an example of an S-function executing information exchange between the SIMULINK model and the Real-Time Kernel. S-functions were developed to give SIMULINK the ability to make generic simulation block to handle in one standard form different roles, like continuous simulation, discrete simulation, system embedding within systems, and so on.

In our applications special S-functions perform information exchange between SIMULINK models and the Real-Time Kernel.

It is important to realise that user-defined S-functions are at the heart of how your control experiments are performed. Inside a proper S-function you can select a real-time controller, set its parameters and make process data being available in MATLAB environment. Notice that this application of SIMULINK models is non-typical**.** In this case **SIMULINK models are applied as a graphical front-end in this case.**

Familiarity with the S-function format is assumed so we shall not go into details about it. If necessary refer to *SIMULINK. DYNAMIC SYSTEM SIMULATION SOFTWARE. USER'S GUIDE*, published by the Mathworks Inc.

Guide for S-functions (example)

We shall discuss the S-function of the *PID Controller* block. This block belongs to the main MS150 window called by the *es* command from the MATLAB Command Window.

The SIMULINK model executed after a double click on *PID Controller* block is shown in Figure 3-1.

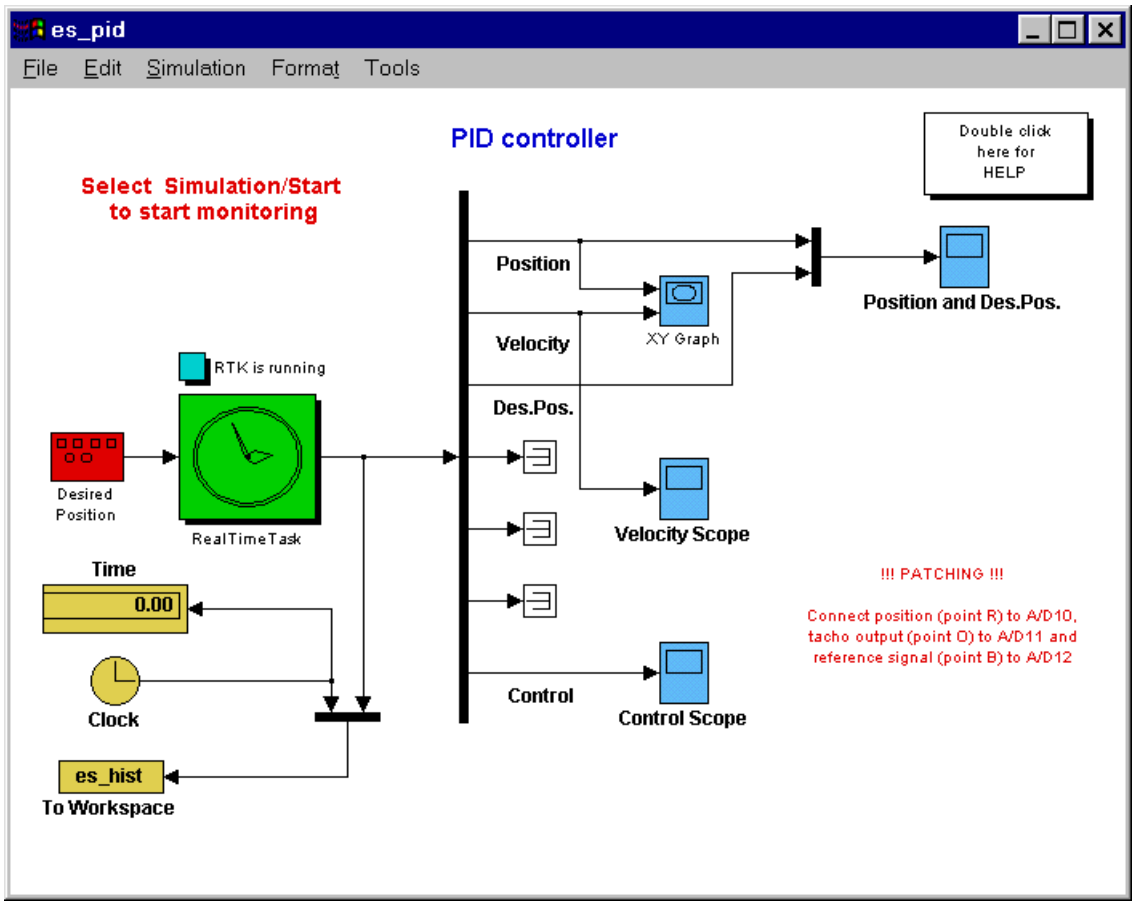


Figure 3-1: *PID Controller* block

Double click on the *Real-Time Task* block and you will see the following dialogue box (Figure 3-2).

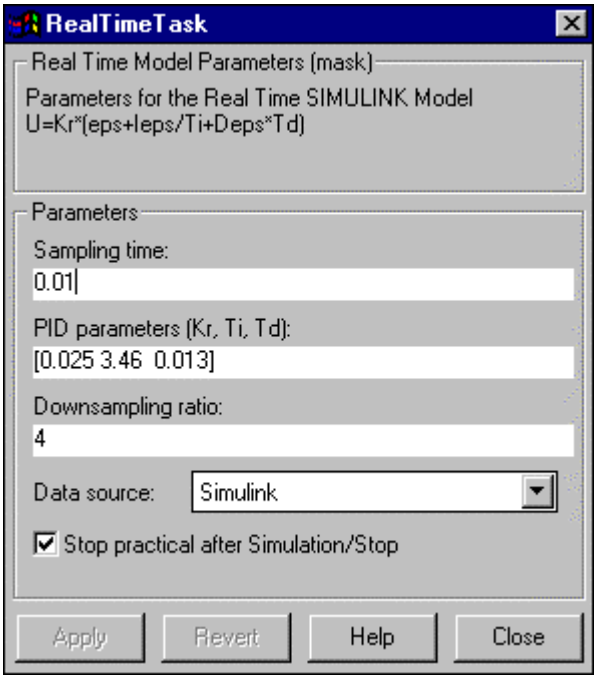


Figure 3-2: Real-Time Task dialogue box.

The window shown in Figure 3-2 sets the parameters of the SIMULINK model. The parameters are passed to the S-function during the execution of the model (after the *Start* command from the Simulation menu).

The *Real-Time Task* block is masked. Masking mechanism, available in SIMULINK, allows you to define a block in terms of its dialogue box, its icon and initialisation commands. To see the details of this block one has to unmask it.

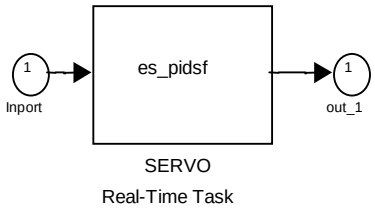


Figure 3-3: Unmasked Real-Time Task block.

The *Real-Time Task* block contains one main block: S-function labelled as *Real-Time Task*.

The blocks *Inport* and *Outport* from Figure 3-3 are required for masking. The name of the S-function connected with the S-function block is *es_pidsf*. This S-function may be found in the *es_pidsf.m* file.

A double click on the *es_pidsf* S-function block opens the following window (Figure 3-4).

Figure 3-4: S-function dialogue box.

In the window shown in Figure 3-4 the user can set the subsystem S-function name and parameters. In this example the function name is *es_pidsf*. The names of the additional function parameters are set during masking of *Real-Time Task* block.

See the *Initialisation* tab field from the *Edit mask* option shown in Figure 3-5.

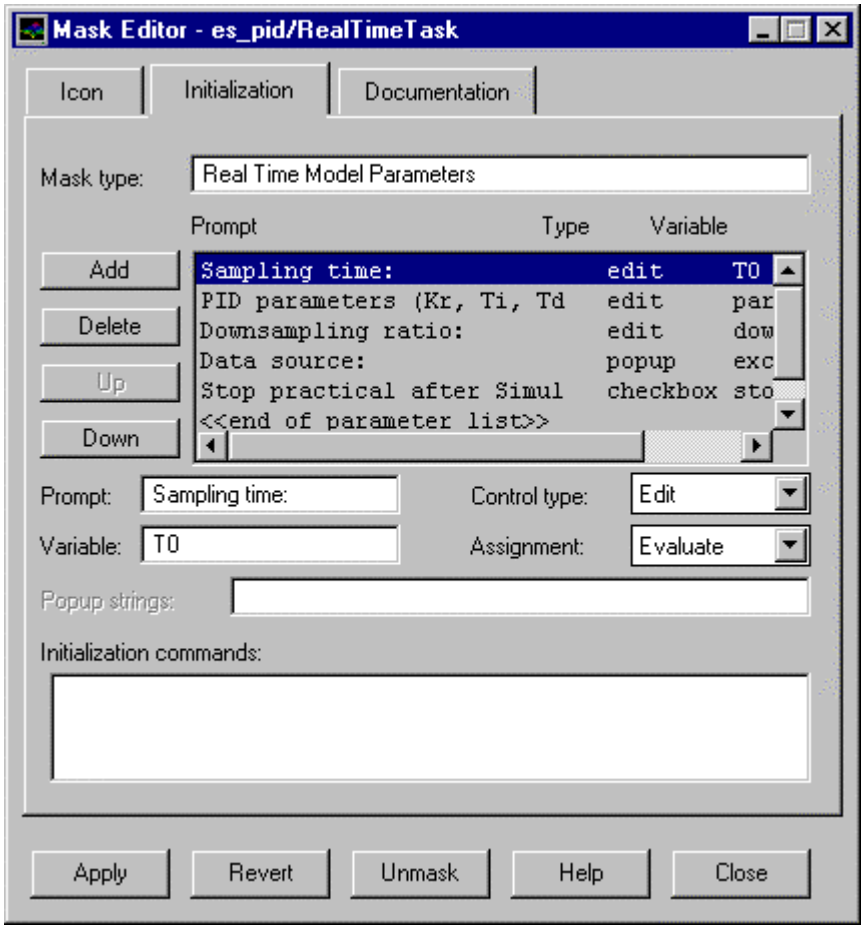


Figure 3-5: Mask definition for the *Real-Time Task* block.

S-function

The content of the *es_pidsf* S-function is described in the following part of this section.

The lines written in italics are statements of the *es_pidsf.m* file. The lines marked on the left side contain comments added for explaining statements of the S-function.

The *es_pidsf* function may use the SIMULINK signal generator, external signal generator or built-in signal source to set the reference angle of the servo disk. The source is selected by the *ext_src variable.*

The *stop_pract* variable is used to stop experiment after the Simulink simulation is terminated. This function reads history of the experiment and transfers samples to the output of S-function block (1 point for every *Downsampling ratio* points).

function [sys, x0, str, ts] = sfunc(t, x, u, flag, T0, par, downsamp, stop_pract, exc_src)

The function has the following parameters:

t	- time,
x	- state vector,
u	- input to the S-function block,
flag	- the value passed to the S-function by SIMULINK to distinguish different actions. The arguments *t*, *x*, *u* and *flag* are set and passed to the S-function by SIMULINK automatically,
par	- parameters for the PID controller,
downsamp	- downsampling ratio. Defines how many samples are transferred to the output of the S-function block. For instance, if *downsamp* is equal to 10 only 1 sample of every 10 samples is transferred from the Real-Time Kernel to the output of the S-function block,
stop_pr	- flag used to stop experiment when the Simulink simulation is terminated,
exc_src	- variable which is used to select the source of the reference position.

Global variables are used to store the history of the experiment and auxiliary variable for downsampling.

```
% Global variables
global  history pos_in_history

switch flag

  case 0, % Initialization
```

Set the number of continuous states, number of discrete states, number of outputs and number of inputs (0 continuous states, 1 discrete state, 7 outputs, 1 input, 0 discontinuous roots, 0 direct feedthrough - without algebraic loops)

```
   sizes.NumContStates  = 0;
   sizes.NumDiscStates  = 1;

   sizes.NumOutputs     = 7;
   sizes.NumInputs      = 1;
   sizes.DirFeedthrough = 0;
```

```
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
```

Set the initial values of the control algorithm. First set the control of the DC motors to zero (control algorithm number 0). Then set the sampling time, set the parameters of the controller and activate control algorithm number 2.

```
dummy = es_call( 'SetAlgNo', 0 );
dummy = es_call( 'SetSampleTime', T0 );

p = es_call( 'GetP' );
Kr = par( 1 );  Ti = par( 2 );  Td = par( 3 );
p( 1 : 5 ) = [ Kr Ti Td 1 1 ];
p = es_call( 'SetP', p );
dummy = es_call( 'SetInitCond', [0 0] );
dummy = es_call( 'SetAlgNo', 2 );  % PID
```

Reset the experiment time

```
dummy = es_call( 'ResetTime', 0 );
```

Set initial state of the internal signal source

```
p = es_call( 'GetPW' );
p( 1 : 2 ) = [ 0 0 ];
p = es_call( 'SetPW', p );
```

Set source of the reference position

```
switch exc_src,
  case 1, % Simulink signal generator
    dummy = es_call( 'SetDataSource', [ 8 9 13 4 5 6 ] );
    dummy = es_call( 'SetPW', [ 0 0 zeros( 1, 18 ) ] );
  case 2, % External signal source
    dummy = es_call( 'SetDataSource', [ 8 9 12 4 5 6 ] );
  case 3, % Built-in signal generator (square)

    dummy = es_call( 'SetDataSource', [ 8 9 13 4 5 6 ] );
    dummy = es_call( 'SetPW', [ 1 3 3 3 3 -50 -50 50 50 zeros( 1, 11 ) ] );
end
```

Wait for the first sample which may be sent to the output

```
while ( es_call( 'GetNoOfSamples', 0 ) <= downsamp )
            ;
    end;

    history = es_call( 'GetHistory', 0 )';

    % initialize the initial conditions
    str = [];       % str is always an empty matrix
    ts  = [-2 0];   % initialize the array of sample times
                    % variable sample time
```

Set the initial value of the *pos_in_history* parameter

```
    % Set initial conditions of the state
    pos_in_history = 1;
    x0 = max( history( :, 1 ) );

    % change background color of Real Time Task block after simulation start
    set_param('es_pid/RealTimeTask','BackgroundColor','cyan');

case 1,  % Unhandled flags
    sys = [];

case 2, % Calculate discrete state
```

If Simulink generator is selected as the source of the reference position set new reference angle

```
    % Set desired value
    if  eq( exc_src, 1 )  % Simulink signal generator
       dummy = es_call( 'SetPW', [ 0 u zeros( 1, 18 ) ] );
    end
    sys = x;

case 3, % Calculate outputs
```

Calculate the output from the S-function block (see body of the *es_sfco.m* file for details)

```
    [ sys, downsamp, history ] = es_sfco( downsamp, history );

case 4, % Calculate next discrete time point
```

Calculate the next discrete time point (see body of the *es_sfntp.m* file for details)

```
    [ sys, downsamp, history ] = es_sfntp( downsamp, history );

case 9, % Terminate
    % change background color of Real Time Task block after simulation stop
    set_param('es_pid/RealTimeTask','BackgroundColor','green');
```

Terminate experiment if the *stop_pract* flag is set

```
    if( stop_pract ~= 0 )
        dummy = es_call( 'SetAlgNo', 0 );
        dummy = es_call( 'SetSampleTime', 0.05 );
    end

otherwise  % Unexpected flags %
    error([ 'Unexpected flag = ' , num2str(flag)] );

end
```

Notes

4 Quick Reference Table

Function Name	Description
1. RTK communication	
GetAlgNo	number of algorithm
GetBaseAddress	get base address of the I/O board
GetChan1Filt, *GetChan2Filt,* *GetChan3Filt,* *GetChan4Filt,* *GetChan5Filt,* *GetChan6Filt*	get parameters of the filters
GetDataSource	get data source parameters
GetDivider	auxiliary clock multiplier
GetHistory	content of the buffer
GetModelP	parameters of the mathematical model
GetNrOfSamples	number of samples in the buffer
GetP	parameters of the control algorithm
GetPW	parameters of internal excitation generator
GetV2Velocity	get conversion coefficient
LoadLibrary	load RTK DLL library
ResetTime	set experiment's time to zero
SetAlgNo	selects the control algorithm and starts the experiment
SetBaseAddress	set base address of MIC 926 board
SetChan1Filt, *SetChan2Filt,* *SetChan3Filt,* *SetChan4Filt,* *SetChan5Filt,* *SetChan6Filt*	set parameters of the filters

SetDataSource	set parameters of the data source selector
SetDivider	set auxiliary clock
SetInitCond	set initial conditions for the mathematical model
SetModelP	set parameters of the mathematical model
SetP	set parameters of the controller
SetPW	set parameters of internal excitation generator
SetSampleTime	set basic clock
SetV2Velocity	set conversion coefficient
StartAcq	clear content of the buffer
StopPractical	stop practical
UnloadLibrary	remove RTK DLL library from memory

2. Demo (Simulink models)	
es	start main demo
es_open	open-loop control
es_pid	PID control algorithm
es_topt	time-optimal control algorithm
es_stflq	state feedback controller
es_mracm	adaptive controller

3. GUI windows	
es_src	define source of input data
es_sad	set base address
es_cpar	set controller parameters
es_egen	set parameters of excitation generator
es_stet	stethoscope GUI window
es_test	basic tests of the hardware
es_tune	calibration of the 33-301 box
es_model	set built-in model oparameters
es_zglr	Ziegler-Nichols method
es_idnt	time domain identification
es_freq	frequency characteristics
es_char	DC drive static characteristic
es_sim	closed-loop system design and simulation

Notes