Twin Rotor MIMO System

Reference Manual

33-007-2M5

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.demon.co.uk   World Wide Web Homepage: http://www.fbk.com

Manual: 33-007-2M5 Ed01 980930   *Printed in England by FI Ltd, Crowborough*

Feedback Part No. 1160–330072M5

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.

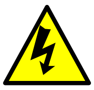
While we provide the fullest possible user information relating to the proper use of this equipment, if there is any doubt whatsoever about any aspect, the user should contact the Product Safety Officer at Feedback Instruments Limited, Crowborough.

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

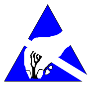
We are required by European Directives to indicate on our equipment panels certain areas and warnings that require attention by the user. These have been indicated in the specified way by yellow labels with black printing, the meaning of any labels that may be fixed to the instrument are shown below:

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

1. Introduction

The software for the TRMS system consists of:

- **Real-Time Kernel (RTK),**
- **Communication functions,**
- **TRMS Toolboxes,**
- **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 (Figure 1-1). The RTK provides a mechanism for real-time measurement and control of the TRMS system in a Windows 95 or Windows NT environment. It has the form of a dynamic linked library (DLL), and is written in a closed form. No modifications are necessary as the RTK contains all the functions which are required for real-time control and data collection. The RTK detects which operating system is running and changes its behaviour according to the operating system environment., and in particular, the RTK starts the kernel-mode device driver which enables access to the I/O address space in the Windows NT environment

Control algorithms are embedded in the real-time kernel, and their selection 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 real-time kernel parameters such as sampling period and excitation source.

Specialised **communication functions** are responsible for communication between the RTK and the MATLAB environment, and these are described in this manual.

The TRMS Toolbox consist of a collection of M-functions in addition to S functions, Simulink models and C-code DLL-files that extend the MATLAB environment for the solution of TRMS modelling, design and control problems. A cyclic memory buffer is created to allow process data flow between the real-time kernel and toolbox functions (MATLAB environment). The TRMS Toolbox, using MATLAB matrix functions and Simulink utilities, provides the user with functions specialised for the real-time control of the twin-rotor MIMO system. It is the general assumption that the toolbox is an **open system** as distinct from the RTK. This approach by its nature makes the basic functions of the toolbox available to the user, and enables the user to create a system of his own and then further customise it to satisfy the requirements better.

However, demonstration M-files for typical TRMS control problems are available in the toolbox and these are described in the TRMS Getting Started manual – 33-007-1M5

The External Interface for the TRMS system offers a 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 External 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** (*RTKIO.SYS*) is required in the Windows NT environment. The driver gives the RTK access to the I/O address space.

The following pages contain the description of the Communication functions. There are 34 Communication functions listed in the alphabetic order. The functions are divided into the four following categories according to the specific roles assigned to them:

- hardware: *GetBaseAddress, SetBaseAddress, ResetEncoders*
- data acquisition: *GetNoOfSamples, ResetTime, GetSampleTime, SetSampleTime, StartAcq, GetHistory*
- software: *LoadLibrary, UnloadLibrary*
- control: all 23 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

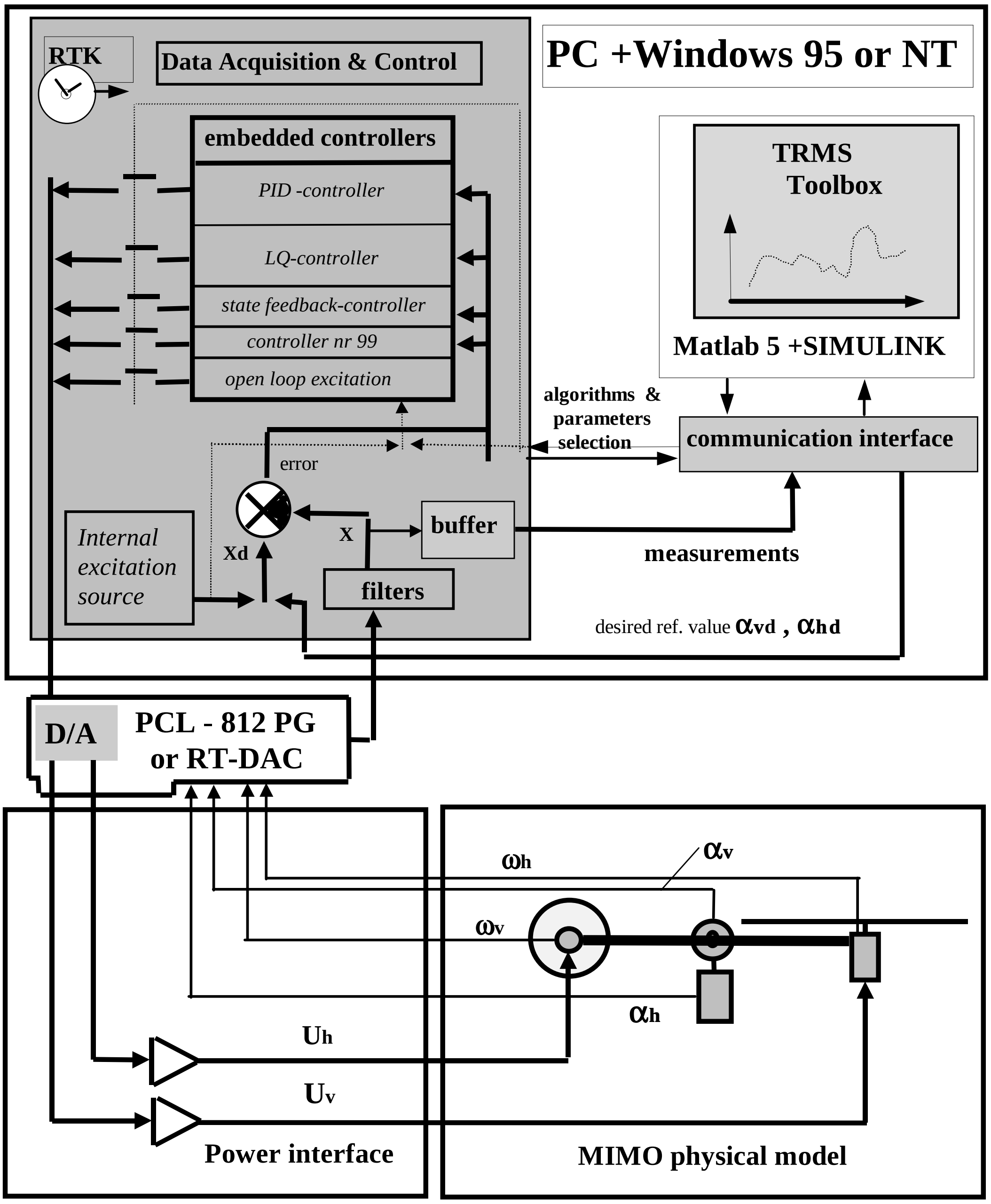


Figure 1-1: Digital control of the TRMS (hardware and software configuration)

Notes

2. Description of the Toolbox Functions

All functions responsible for communication between the Real-Time Kernel (RTK) and MATLAB environment are in the following form:

ReturnValue = hl_call(FunctionName, [Argument])

where:

ReturnValue - value returned by the function,

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

FunctionName - name of the desired operation, string format,

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

2.1 GetAlgNo

Purpose: Get number of the currently active control algorithm.

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

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

See: *SetAlgNo*

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

```
function dspdsr

  AlgNo = hl_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( 'Extended LQ controller' );
  elseif AlgNo == 99
    disp( 'External controller' );
  else
    disp( '!!! Unknown controller !!!' );
  end
```

2.2 GetBaseAddress

Purpose: Get base address of I/O board.

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

Description: The function is called to obtain the base address of PCL-812PG or the RT-DAC interface board.

See*:* *SetBaseAddress, GetModelP, SetModelP, SeInitCond*

Note*:* If the base address returned by the *GetBaseAddress* function is equal to zero the RTK generates data from the mathematical model of the TRMS system. See description of the *GetModelP*, *SetModelP* and *SetInitCond* functions for details.

Example 1*:* *BaseAddr = hl_call('GetBaseAddress')*
BaseAddr =

544

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 following commands display the example of data generated by the built-in mathematical model.

ret = hl_call('SetBaseAddress', 0);

ret = hl_call('StartAcq'); pause(5)

hist = hl_call('GetHistory');

plot(hist(1, :), hist([2 4 6], :))

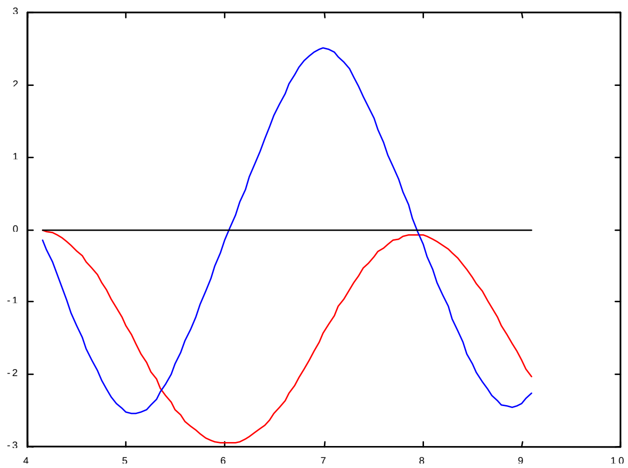


Figure 2-1: Data generated by the mathematical model.

2.3 GetDivider

Purpose: Get divider of the auxiliary clock.

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

Description: The function returns the divider of the basic RTK clock

See: *SetDivider, GetSampleTime, SetSampleTime*

2.4 GetHistory

Purpose: Get content of the internal RTK buffer.

Synopsis*:* *Hist = hl_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:

Table 1.

Row of the matrix	Description	Units
Hist(1,:)	regularly spaced time	seconds
Hist(2,:)	vertical angle	radians
Hist(3,:)	horizontal angle	radians
Hist(4,:)	vertical angular velocity	radians/sec
Hist(5,:)	horizontal angular velocity	radians/sec
Hist(6,:)	velocity of the vertical rotor	A/D units
Hist(7,:)	velocity of the horizontal rotor	A/D units
Hist(8,:)	vertical control	relative units in the range ±1
Hist(9,:)	horizontal control	relative units in the range ±1
Hist(10,:)	vertical desired angle	radians
Hist(11,:)	horizontal desired angle	radians
Hist(12,:) [*)]	data from force sensor	A/D units

[*)] - a special force sensor is required.

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.

See: *GetNoOfSamples, StartAcq*

Note: The GUI tool is available which displays data collected by the RTK. Execute the *hl_stet* command in the MATLAB command window to call this tool (see Figure 2-2).

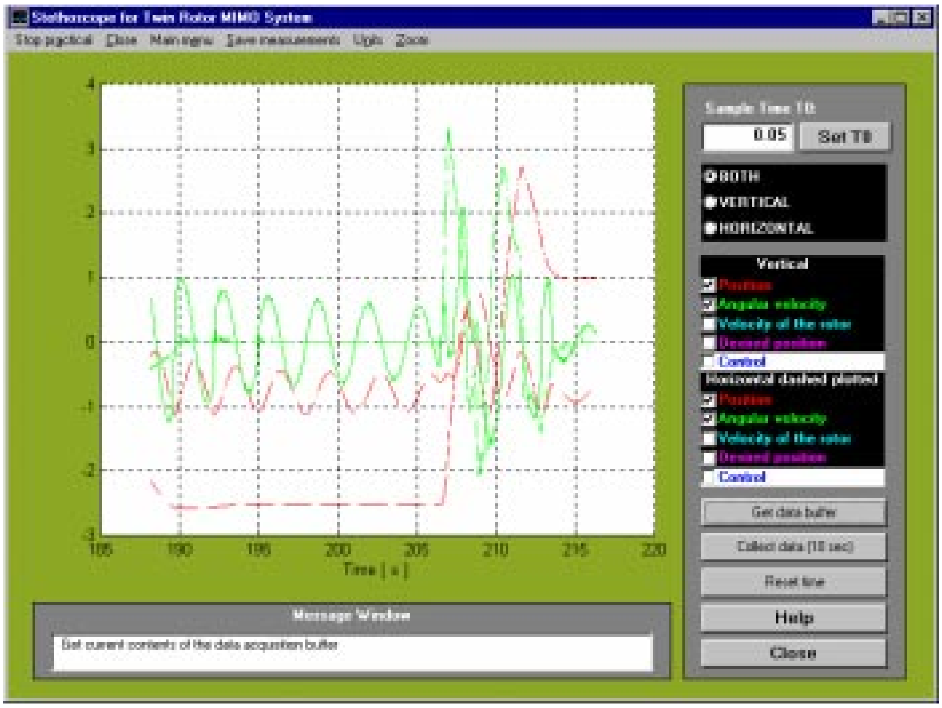


Figure 2-2: The result of the *hl_stet* command.

Example 1: Read 250 data points and display the data. All data presented in this example were obtained during PID control experiment.

hl_call('StartAcq'); *% clear data buffer*

while(hl_call('GetNoOfSamples') < 250) ; *% wait for data*

end;

Hist = hl_call('GetHistory'); *% get data and*

% clear buffer

To plot the angle of the vertical angle of the beam, execute the following command:

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

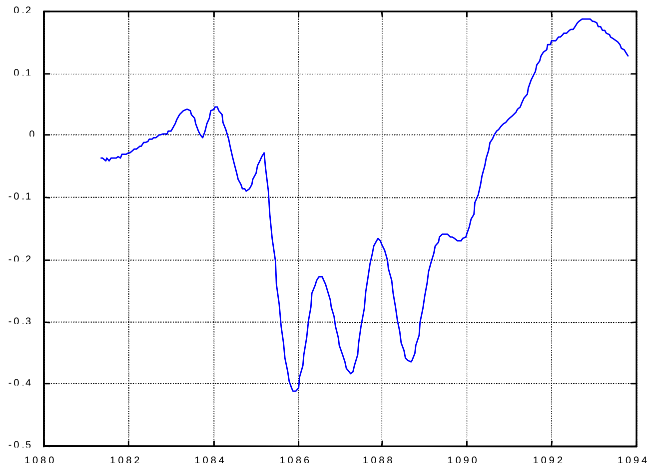


Figure 2-3: Vertical angle vs. time.

Figure 2-3 shows the vertical angle vs. time diagram. Notice that time starts from approximately 1081 seconds. This means that the experiment is started after 1081 seconds after the moment when *hl_call* library was loaded to the memory. If you want to plot the time from zero, modify the command :

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

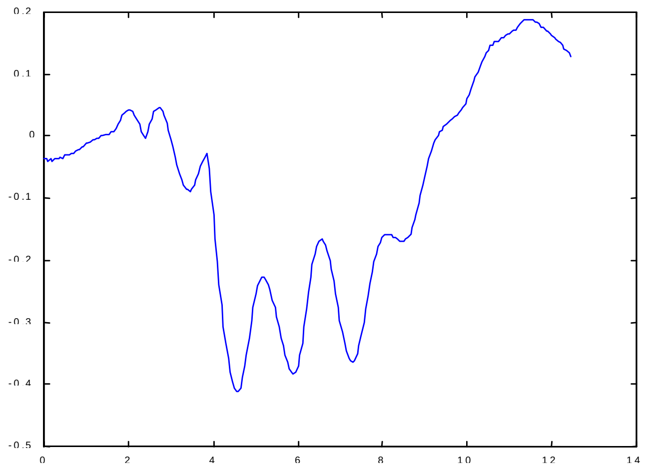


Figure 2-4: Vertical angle vs. time (plot started from zero).

Figure 2-4 shows the plot.

The following commands plot vertical angle and vertical angular velocity vs. time (see Figure 2-5):

plot(Hist(1, :), Hist(2, :), Hist(1, :), Hist(4, :))

Figure 2-5: Vertical angle and vertical angular velocity vs. time.

To show velocity of the horizontal rotor execute the following command (see Figure 2-6):

plot(Hist(1, :), Hist(6, :)); grid

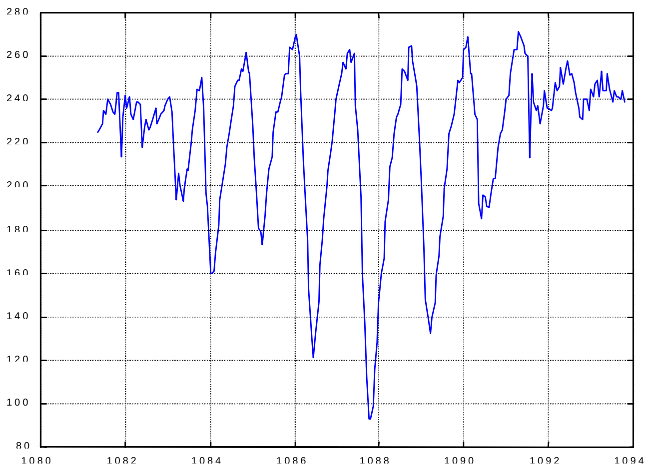


Figure 2-6: Velocity of the horizontal rotor vs. time.

The following command plots control value for both DC drives (Figure 2-7):

plot(Hist(1, :), Hist(8, :), Hist(1, :), Hist(9, :))

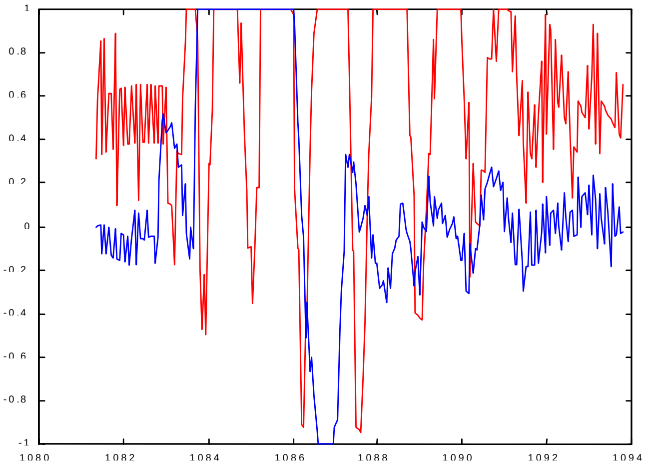


Figure 2-7: Control vs. time.

2.5 GetHorizPosFilter

Purpose: Get the parameters of the horizontal position digital filter.

Synopsis: *fi = hl_call('GetHorizPosFilter')*

Description: The function gets the parameters of the digital filter currently active in the RTK. The filter associated with the *GetHorizPosFilter* function operates on the data describing the horizontal angle of the beam.

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 the 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).

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: *SetHorizPosFilter, GetHorizRotorFilter, GetHorizSpeedFilter*

2.6 GetHorizRotorFilter

Purpose: Get the parameters of the digital filter for the horizontal rotor velocity.

Synopsis*:* *fi = hl_call('GetHorizRotorFilter')*

Description: The function gets the parameters of the digital filter currently active in the RTK. The filter associated with the *GetHorizRotorFilter* function operates on the velocity data of the main rotor.
See description of the *GetHorizPosFilter* for filter details.

See*:* *SetHorizRotorFilter, GetHorizPosFilter, GetHorizSpeedFilter*

2.7 GetHorizSpeedFilter

Purpose: Get the parameters of the horizontal velocity digital filter.

Synopsis*:* *fi = hl_call('GetHorizSpeedFilter')*

Description: The function gets the parameters of the digital filter currently active in the RTK. The filter associated with the *GetHorizSpeedFilter* function operates on the horizontal velocity data of the beam.
See description of the *GetHorizPosFilter* for filter details.

See*:* *SetHorizSpeedFilter*, *GetHorizPosFilter*, *GetHorizRotorFilter*

2.8 GetModelP

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

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

Description: The function returns the 8x6 matrix which contains parameters of the mathematical model of the TRMS. The elements of the matrix correspond to the mathematical model of the real system and are given by the equations shown below (see *Advanced Teaching Manual 1* for details and physical interpretation of all parameters).

The state equations of the model are:

$$\frac{dS_v}{dt} = \frac{l_m F_v(\omega_m) S_f - \Omega_v k_v + g((A-B)\cos\alpha_v - C\sin\alpha_v) - \frac{1}{2}\Omega_h^2 (A+B+C)\sin 2\alpha_v}{J_v},$$

$$\frac{d\alpha_v}{dt} = \Omega_V = \frac{S_v + J_{tr}\,\omega_t}{J_v},$$

$$\frac{dS_h}{dt} = \frac{l_t F_h(\omega_t) S_f \cos\alpha_v - \Omega_h k_h}{J_h},$$

$$\frac{d\alpha_h}{dt} = \Omega_h = \frac{S_h + J_{mr}\,\omega_m \cos\alpha_v}{J_h},$$

$$\frac{du_{vv}}{dt} = \frac{1}{T_{mr}}(-u_{vv} + u_v),$$

$$\frac{du_{hh}}{dt} = \frac{1}{T_{tr}}(-u_{hh} + u_h),$$

where:

$$\omega_m(u_{vv}) = \sum_{i=0}^{7} p_1(i) u_{vv}^i,$$

$$F_v(\omega_m) = \sum_{i=0}^{7} p_2(i) \omega_m^i,$$

$$\omega_t(u_{hh}) = \sum_{i=0}^{7} p_3(i) u_{hh}^i,$$

$$F_h(\omega_t) = \sum_{i=0}^{7} p_4(i) \omega_t^i,$$

$$A = \left(\frac{m_t}{2} + m_{tr} + m_{ts} \right) l_t,$$

$$B = \left(\frac{m_m}{2} + m_{mr} + m_{ms} \right) l_m,$$

$$C = \left(\frac{m_b}{2} l_b + m_{cb} l_{cb} \right),$$

$$D = \frac{m_b}{3} l_b^2 + m_{cb} l_{cb}^2,$$

$$E = \left(\frac{m_m}{3} + m_{mr} + m_{ms} \right) l_m^2 + \left(\frac{m_t}{3} + m_{tr} + m_{ts} \right) l_t^2,$$

$$F = m_{ms} r_{ms}^2 + \frac{m_{ts}}{2} r_{ts}^2,$$

$$J_h = D \cos^2\alpha_v + E \sin^2\alpha_v + F,$$

S_f = 5/(2.895*2048), - constant scaling factor,

where:

S_v, S_h are moments of momentum of the beam in both axis (auxiliary variables),

α_v is the pitch angle of the beam,

ω_m is angular velocity of the main rotor,

Ω_h is the angular velocity of the beam around the vertical axis,

α_h is the azimuth angle of the beam,

Ω_v is the angular velocity around the horizontal axis,

ω_t is angular velocity of the tail rotor,

$p_1(i)$, $p_2(i)$, $p_3(i)$, $p_4(i)$ are constant coefficients of the polynomials,

u_v, u_h control for the main and tail propeller,

m_{mr} is the mass of the main DC-motor with main rotor,

m_m is the mass of main part of the beam,

m_{tr} is the mass of the tail motor with tail rotor,

m_t is the mass of the tail part of the beam,

m_{cb} is the mass of the counter-weight,

m_b is the mass of the counter-weight beam,

m_{ms} is the mass of the main shield,

m_{ts} is the mass of the tail shield,

l_m is the length of main part of the beam,

l_t is the length of tail part of the beam,

l_b is the length of the counter-weight beam,

l_{cb} is the distance between the counter-weight and the joint,

r_{ms} is radius of main shield,

r_{ts} is radius of tail shield,

k_v, k_h are a constants,

J_v is the sum of moments of inertia relative to the horizontal axis,

J_h is the sum of moments of inertia relative to the vertical axis,

T_{mr} time constant of main motor- propeller system,

T_{tr} time constant of tail motor- propeller system,

J_{tr} moment of inertia in DC-motor -tail propeller subsystem,

J_{mr} moment of inertia in DC-motor -main propeller subsystem,

$F_v(\omega_m)$ denotes the dependence of the propulsive force on the angular velocity of the rotor,

$F_h(\omega_t)$ denotes the dependence of the propulsive force on the angular velocity of the tail rotor,

g is gravitational acceleration.

The output from the simulation function is the vector $Y = \begin{bmatrix} \alpha_v \\ \alpha_h \\ \Omega_v \\ \Omega_h \\ \omega_m \\ \omega_t \end{bmatrix}$. The vector may be stored in the RTK data acquisition buffer.

The parameters are stored in an 8x6 matrix. The contents of the elements of the matrix is shown in the table below:

A	J_{mr}	$p_1(0)$	$p_2(0)$	$p_3(0)$	$p_4(0)$
B	D	$p_1(1)$	$p_2(1)$	$p_3(1)$	$p_4(1)$
C	E	$p_1(2)$	$p_2(2)$	$p_3(2)$	$p_4(2)$
J_v	F	$p_1(3)$	$p_2(3)$	$p_3(3)$	$p_4(3)$
l_m	l_t	$p_1(4)$	$p_2(4)$	$p_3(4)$	$p_4(4)$
k_v	k_h	$p_1(5)$	$p_2(5)$	$p_3(5)$	$p_4(5)$
g	T_{mr}	$p_1(6)$	$p_2(6)$	$p_3(6)$	$p_4(6)$
J_{tr}	T_{tr}	$p_1(7)$	$p_2(7)$	$p_3(7)$	$p_4(7)$

The default values of the parameters are:

0.0217	2.65e-5	0.0	0.0	0.0	0.0
0.0119	0.049	1283.41	9.544e-2	3796.83	0.801
0.01678	0.0016	63.45	-1.632e-4	262.27	-1.808e-4
0.1099	0.00633	-1238.64	4.123e-6	-4283.15	2.511e-7
0.236	0.25	-129.26	1.09e-9	-194.69	-1.595e-11
0.00545	0.0095	599.73	-3.48e-12	2020.0	-3.0e-14
9.81	1.432	90.99	0.0	0.0	0.0
1.654e-5	0.3842	0.0	0.0	0.0	0.0

The built-in mathematical model is used to calculate values of the outputs of the system when the base address of the input/output board is set to zero. In such a case the values obtained from the mathematical model (instead of measured data) are stored in the data acquisition buffer.

See: *SetModelP, SetInitCond, GetBaseAddress*

2.9 GetNoOfSamples

Purpose: Get number of samples available in the buffer.

Synopsis*:* *Hist = hl_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.

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

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

```
while( hl_call( 'GetNoOfSamples' ) < 1200 ) ;
end;
```

2.10 GetP

Purpose: Get parameters of the control algorithm.

Synopsis*:* *Par = hl_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 parameters is common for all algorithms, but only specified values are used by a currently active controller. See the *SetAlgNo* function for details.

See*:* *GetPW*, *SetP*

Example*:* Increase the second parameter by 120%

Par = hl_call('GetP');

*Par(2) = 1.2 * Par(2);*

hl_call('SetP', Par);

2.11 GetPWHoriz

Purpose: Get the parameters of the internal excitation source for horizontal axis.

Synopsis*:* *Par = hl_call('GetPWHoriz')*

Description: The function returns the vector containing the parameters of the internal signal generator for horizontal axis. The vector contains 10 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 available in the RTK may be used as:

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

See*:* *SetPWHoriz*

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

Par = zeros(1, 10);

Par(1 : 5) = [1 -1.2 4.66 0 7];

hl_call('SetPWHoriz', Par);

Par = hl_call('GetPWHoriz')

Par =

1 -1.2 4.66 0 7 0 0 0 0 0

2.12 GetPWVert

Purpose: Get the parameters of the internal excitation source for vertical axis.

Synopsis*:* *Par = hl_call('GetPWVert')*

Description: The function returns the vector containing the parameters of the internal signal generator for vertical axis. The vector contains 10 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 drive in the open-loop mode,
- a source of reference angle in the closed-loop modes.

See*:* *GetPWHoriz, SetPWVert*

2.13 GetSampleTime

Purpose: Get basic sampling time.

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

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

See*:* *SetSampleTime*

2.14 GetVertPosFilter

Purpose: Get the parameters of the vertical position digital filter.

Synopsis*:* *fi = hl_call('GetVertPosFilter')*

Description: The function gets the parameters of the digital filter currently active in the RTK. The filter associated with the *GetVertPosFilter* function operates on the vertical position data of the beam.

See description of the *GetHorizPosFilter* for filter details.

See*:* *SetVertPosFilter, GetVertRotorFilter, GetVertSpeedFilter*

2.15 GetVertRotorFilter

Purpose: Get the parameters of the digital filter for the vertical rotor velocity.

Synopsis*:* *fi = hl_call('GetVertRotorFilter')*

Description: The function gets the parameters of the digital filter currently active in the RTK.

The filter associated with the *GetVertRotorFilter* function operates on the velocity data of the tail rotor.

See description of the *GetHorizPosFilter* for filter details.

See*:* *SetVertRotorFilter, GetVertPosFilter, GetVertSpeedFilter*

2.16 GetVertSpeedFilter

Purpose: Get the parameters of the vertical velocity digital filter.

Synopsis*:* *fi = hl_call('GetVertSpeedFilter')*

Description: The function gets the parameters of the digital filter currently active in the RTK.

The filter associated with the *GetVertSpeedFilter* function operates on the vertical velocity data of the beam.

See description of the *GetHorizPosFilter* for filter details.

See*:* *SetVertSpeedFilter, GetVertPosFilter, GetVertRotorFilter*

2.17 LoadLibrary

Purpose: Load RTK to memory.

Synopsis: *ret = hl_call('LoadLibrary')*

Description: The function loads the RTK to the memory. The RTK is a part of the *hl_call.dll* executable.

First call to any *hl_call* function loads the RTK module into memory automatically. So in fact, the explicit call to the *LoadLibrary* function is not necessary.

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

Can not open HL_PAR.INI file. Base address is set to zero - the *hl_par.ini* file is not present in the root MATLAB directory. The *hl_par.ini* file should contain the following line which defines base address of the input/output board:

BaseAddress= 544

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

Can not find BaseAddress parameter in the HL_PAR.INI file - the *hl_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 running in the operating system. See the description of installation procedure to solve this problem.

See: UnloadLibrary

2.18 ResetEncoders

Purpose: Set the incremental encoders to the initial position.

Synopsis*:* *hl_call('ResetEncoders')*

Description: The function sets the initial position of the TRMS system. The function is called while the system is in a steady-state and sets "zero" position of the beam.

The *ResetEncoders* function should be executed immediately after loading RTK to the memory.

See*:* *LoadLibrary*

2.19 ResetTime

Purpose: Set the experiment time to zero.

Synopsis*:* *hl_call('ResetTime')*

Description: The function sets the time counter of the RTK to zero. Time is calculated from the first call of the *hl_call* function.

After a call of the *ResetTime* function the first row of the matrix returned by the *GetHistory* function starts from zero.

See*:* *GetHistory*

2.20 SetAlgNo

Purpose: Select a control algorithm and start the experiment.

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

Description: The function is called to select a control algorithm 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*, *SetPWHoriz* or *SetPWVert* functions. Use the *SetPWHoriz* and the *SetPWVert* functions before *hl_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 generators as control signal sources
2	PID controller
3	PID controller with feedback from rotor velocity
4	State feedback
99	External controller

See: *GetAlgNo, GetP, GetPWHoriz, GetPWVert, SetP, SetPWHoriz, SetPWVert*

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

```
hl_call( 'SetAlgNo', 0 );                     % stop previous            %
experiment
hl_call( 'SetPWHoriz', [ 2 5 1 -0.7 0.5] );   % set triangle excitation
                                              % for horizontal
                                              % position
hl_call( 'SetPWVert', [ 2 5 1 -1.0 1.0] );    % set triangle excitation
                                              % for vertical position
hl_call( 'SetAlgNo', 1 );                     % start experiment
hl_call( 'GetAlgNo' )
ans =
        1
```

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

```
hl_call( 'SetAlgNo', 0 );          % stop experiment
hl_call( 'SetP', [                 1.774   0.0002 ... % Kpvv and Kpvh
                        0.5   0 ...      % Kivv and Kivh
                       1.82   0.1 ...    % Kdvv and Kdvh
                       0.1   0.298 ...   % Kphv and Kphh
                       0   0.303  ...    % Kihv and Kihh
                       0   0.346...      % Kdhv and Kdhh
                       1.194   0.2 ...   % Ivvsat and Ivhsat
                       0.2   0.798 ...   % Ihvsat and Ihhsat
                       0.583 0.175 ] );               % Uvmax and Uhmax
hl_call( 'SetPWHoriz', [ 2 5 1 -0.7 0.5] );   % set triangle reference
                                              % position wave
hl_call( 'SetPWVert', [ 2 5 1 -1.0 1.0] );    % set triangle reference
                                              % position wave
hl_call( 'SetAlgNo', 2 );          % begin PID experiment
```

2.21 SetBaseAddress

Purpose: Set base address of the PCL-812PG or the RT-DAC board.

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

Description: The function is called to set a new base address of the PCL-812PG or RT-DAC boards.

Notice, that the base address is fixed in the configuration file *hl_par.ini* and is read each time when the RTK is loaded to memory. It the address of the I/O board is set to 0 the RTK generates date from mathematical model of the system (see the description of the *SetModelP*, *GetModelP* and *SetInitCond* functions).

See*:* *GetBaseAddress*

Notes*:* 1. There is a GUI tool available which can be used to set the base address of the I/O board. The tool is called by the *hl_saddr* command. The view of the tool is shown below.

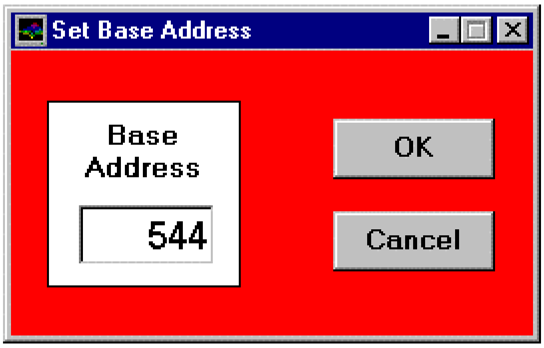


Figure 2-8: The result of the *hl_saddr* command.

2. The base address is loaded from the *hl_par.ini* file during loading the RTK library to the memory. Change the appropriate line in this file to change base address permanently.

The *GetBaseAddress* command sets the new value of the base address only in the current session.

3. If the *hl_par.ini* file contains the following line:

RT-DAC

the RTK co-operates with the RT-DAC I/O board. If such line is absent the RTK works with the PCL-812PG card.

Example: Set the address of the PCL-812 board to 544 (220 in hexadecimal code).

```
hl_call( 'SetBaseAddress', 544 );
hl_call( 'GetBaseAddress' )
ans =
     544
```

The following command causes generation of "dummy" data by the RTK:

```
hl_call( 'SetBaseAddress', 0 );
```

2.22 SetDivider

Purpose: Set the clock divider.

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

Description: The function sets an auxiliary clock of RTK. This function allows to define a control period different from the value set by *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.

hl_call('SetDivider', 3);

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

hl_call('SetDivider', 1);

Example 3: The following commands:

```
% set max value od control
hl_call( 'SetP', [ 1 zeros( 1, 19 ) ] );
% select open-loop control
hl_call( 'SetAlgNo',1 );
hl_call( 'SetDivider', 1 );
% set triangle open-loop excitation
hl_call( 'SetPWHoriz', ...
          [2 1 2 -1 1 zeros( 1, 15 ) ] );
hl_call( 'SetSampleTime', 0.1 );
h = hl_call( 'StartAcq' );
pause( 5 )
h = hl_call( 'GetHistory' );
stairs( h( 1, : )-h( 1, 1 ),h( 6, : ) )
```

create the following diagram:

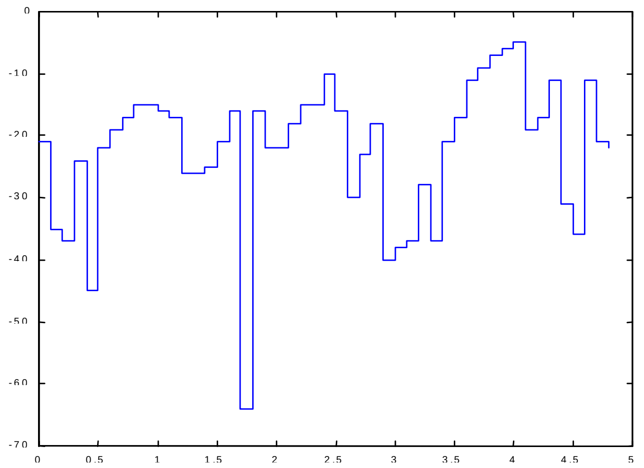


Figure 2-9: Results for divider equal to 1.

After the command

```
hl_call( 'SetDivider', 5 );
```

the picture changes (see Figure 2-10). Notice that the control value changes three times slower.

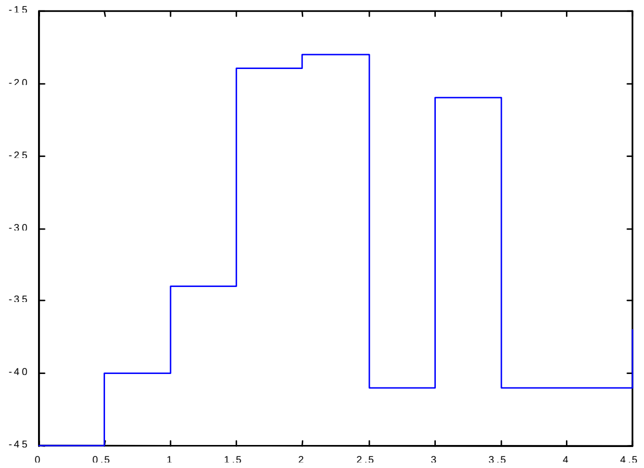


Figure 2-10: Results for divider equal to 5.

2.23 SetHorizPosFilter

Purpose: Set the parameters of the horizontal position digital filter.

Synopsis*:* *ret = hl_call('SetHorizPosFilter', fi)*

Description: The function sets the parameters of the digital filter currently active in the RTK. The filter associated with the *GetHorizPosFilter* function operates on the data describing the horizontal angle of the beam.
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 another 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*:* *GetHorizPosFilter, SetHorizRotorFilter, SetHorizSpeedFilter*

2.24 SetHorizRotorFilter

Purpose: Set the parameters of the digital filter for the horizontal rotor velocity.

Synopsis*:* *ret = hl_call('SetHorizRotorFilter', fi)*

Description: The function sets the parameters of the digital filter currently active in the RTK. The filter associated with the *SetHorizRotorFilter* function operates on the velocity data of the main rotor.

See description of the *SetHorizPosFilter* for filter details.

See*:* *GetHorizRotorFilter, SetHorizPosFilter, SetHorizSpeedFilter*

2.25 SetHorizSpeedFilter

Purpose: Set the parameters of the horizontal velocity digital filter.

Synopsis*:* *ret = hl_call('GetHorizSpeedFilter', fi)*

Description: The function sets the parameters of the digital filter currently active in the RTK.

The filter associated with the *SetHorizSpeedFilter* function operates on the horizontal velocity data of the beam.

See description of the *SetHorizPosFilter* for filter details.

See*:* *GetHorizSpeedFilter, SetHorizPosFilter, SetHorizRotorFilter*

2.26 SetInitCond

Purpose: Set the initial conditions for the mathematical model.

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

Description: The function sets the initial conditions for the mathematical model built into the RTK. New initial conditions are applied starting from the next sample time step.

The *par* vector contains six elements:

par(1) is initial value of the S_v variable,

par(2) is initial value for the α_v,

par(3) is initial value for the S_h,

par(4) is initial value for the α_h,

par(5) is initial value for the u_{vv} and

par(6) is initial value for the u_{hh}.

See the *Advanced Teaching Manual 1* for details of the model equations.

See*:* *GetModelP, SetModelP*

2.27 SetModelP

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

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

Description: The function sets the parameters of the mathematical model. The *par* input argument is a 8x6 matrix which contains parameters of the model.

See description of the *GetModelP* function for details of the model.

See: *GetModelP, SetInitCond*

2.28 SetP

Purpose: Set the parameters of the controller.

Synopsis: *hl_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 in the *Par* input argument have the following meaning:

For algorithm number 0:

the parameters are not required,

For algorithm number 1:

see function *SetPWHoriz* and *SetPWHori* for detailed description,

For algorithm number 2 (PID controller):

the controller is described by the equations given below.

$$\varepsilon_v = \alpha_{vd} - \alpha_v,$$

$$\varepsilon_h = \alpha_{hd} - \alpha_h,$$

where:

$\varepsilon_v, \varepsilon_h$	are errors of vertical and horizontal angle,
α_{vd}, α_{hd}	are reference values of vertical and horizontal angles,
α_v, α_h	are vertical and horizontal angles.

The integrators are described by the following equations:

$$I_{vv}(t) = K_{ivv}\int_0^t \varepsilon_v dt,$$

$$if\,(I_{vv} > I_{vvsat})\,then\,I_{vv} = I_{vvsat},\; if\,(I_{vv} < -I_{vvsat})\,then\,I_{vv} = -I_{vvsat},$$

$$I_{vh}(t) = K_{ivh}\int_0^t \varepsilon_v dt,$$

$$if\,(I_{vh} > I_{vhsat})\,then\,I_{vh} = I_{vhsat},\; if\,(I_{vh} < -I_{vhsat})\,then\,I_{vh} = -I_{vhsat},$$

$$I_{hv}(t) = K_{ihv}\int_0^t \varepsilon_h dt,$$

$$if\,(I_{hv} > I_{hvsat})\,then\,I_{hv} = I_{hvsat},\; if\,(I_{hv} < -I_{hvsat})\,then\,I_{hv} = -I_{hvsat},$$

$$I_{hh}(t) = K_{ihh}\int_0^t \varepsilon_h dt,$$

$$if\,(I_{hh} > I_{hhsat})\,then\,I_{hh} = I_{hhsat},\; if\,(I_{hh} < -I_{hhsat})\,then\,I_{hh} = -I_{hhsat},$$

where:

$K_{ivv}, K_{ivh}, K_{ihv}, K_{ihh}$ are gains of the I parts of the controller,

$I_{vvsat}, I_{vhsat}, I_{hvsat}, I_{hhsat}$ are saturation levels of the integrators.

Finally, vertical and horizontal controls are:

$$\boxed{U_v = K_{pvv}\varepsilon_v + I_{vv}(t) + K_{dvv}\frac{d\varepsilon_v}{dt} + K_{pvh}\varepsilon_h + I_{vh}(t) + K_{dvh}\frac{d\varepsilon_h}{dt},}$$

$$if\,(U_v > U_{v\max})\,then\,U_v = U_{v\max},\; if\,(U_v < -U_{v\max})\,then\,U_v = -U_{v\max},$$

$$\boxed{U_h = K_{phv}\varepsilon_h + I_{hv}(t) + K_{dhv}\frac{d\varepsilon_v}{dt} + K_{phh}\varepsilon_h + I_{hh}(t) + K_{dhh}\frac{d\varepsilon_h}{dt},}$$

$$if\,(U_h > U_{h\max})\,then\,U_h = U_{h\max},\; if\,(U_h < -U_{h\max})\,then\,U_h = -U_{h\max},$$

where:

$K_{pvv}, K_{pvh}, K_{phv}, K_{phh}, K_{dvv}, K_{dvh}, K_{phv}, K_{phh}$ are parameters of the controller,

$U_{v\max}, U_{h\max}$ are the limits of the vertical and horizontal control values.

The positions of the parameters from the *Par* input argument are shown in the table below.

Table 3. Position of the parameter for the PID controller.

Position	Parameter	Position	Parameter
1	K_{Pvv}	10	K_{Ihh}
2	K_{Pvh}	11	K_{Dhv}
3	K_{Ivv}	12	K_{Dhh}
4	K_{Ivh}	13	I_{vvsat}
5	K_{Dvv}	14	I_{vhsat}
6	K_{Dvh}	15	I_{hvsat}
7	K_{Phv}	16	I_{hhsat}
8	K_{Phh}	17	$U_{v\max}$
9	K_{Ihv}	18	$U_{h\max}$

For algorithm number 3 (state-feedback controller):

The control outputs from the state-feedback controller are calculated according to the following formulas:

$$\varepsilon_v = \alpha_{vd} - \alpha_v,$$

$$\varepsilon_h = \alpha_{hd} - \alpha_h,$$

$$\boxed{U_v = K_{v1}\varepsilon_v + K_{v2}\Omega_v + K_{v3}\varepsilon_h + K_{v4}\Omega_h,}$$

$$if\,(U_v > U_{v\max})\,then U_v = U_{v\max},\;\; if\,(U_v < -U_{v\max})\,then U_v = -U_{v\max},$$

$$\boxed{U_h = K_{h1}\varepsilon_v + K_{h2}\Omega_v + K_{h3}\varepsilon_h + K_{h4}\Omega_h,}$$

$$if\,(U_h > U_{h\max})\,then U_h = U_{h\max},\;\; if\,(U_h < -U_{h\max})\,then U_h = -U_{h\max},$$

where:

$\varepsilon_v, \varepsilon_h$ are errors of vertical and horizontal angle,

α_{vd}, α_{hd} are reference values of vertical and horizontal angle,

α_v, α_h are vertical and horizontal angles,

Ω_v, Ω_h are angular velocities with respect to the vertical and horizontal axis,

$K_{v1},\ldots,K_{v4}, K_{h1},\ldots,K_{h4}$ are parameters of the controller,

$U_{v\max}, U_{h\max}$ are maximum values of the vertical and horizontal control.

The positions of the parameters are shown in the table below.

Table 4. Position of the parameter for the state-feedback controller.

Position	Parameter	Position	Parameter
1	K_{v1}	6	K_{h2}
2	K_{v2}	7	K_{h3}
3	K_{v3}	8	K_{h4}
4	K_{v4}	9	$U_{v\max}$
5	K_{h1}	10	$U_{h\max}$

For algorithm number 4 (extended state-feedback controller):

The control outputs from the state-feedback controller are calculated according to the following formulas:

$$\varepsilon_v = \alpha_{vd} - \alpha_v,$$

$$\varepsilon_h = \alpha_{hd} - \alpha_h,$$

$$\boxed{U_v = K_{v1}\varepsilon_v + K_{v2}\Omega_v + K_{v3}\Omega_{ROTv} + K_{v4}\varepsilon_h + K_{v5}\Omega_h + K_{v6}\Omega_{ROTh} + K_{v7},}$$

$$if\,(U_v > U_{v\max})\,then U_v = U_{v\max},\;\; if\,(U_v < -U_{v\max})\,then U_v = -U_{v\max},$$

$$\boxed{U_h = K_{h1}\varepsilon_v + K_{h2}\Omega_v + K_{h3}\Omega_{ROTv} + K_{h4}\varepsilon_h + K_{h5}\Omega_h + K_{h6}\Omega_{ROTh} + K_{h7},}$$

$$if\,(U_h > U_{h\max})\,then U_h = U_{h\max},\;\; if\,(U_h < -U_{h\max})\,then U_h = -U_{h\max},$$

where:

$\varepsilon_v, \varepsilon_h$ are errors of vertical and horizontal angle,

α_{vd}, α_{hd} are reference values of vertical and horizontal angle,

α_v, α_h are vertical and horizontal angles,

Ω_v, Ω_h are angular velocities with respect to the vertical and horizontal axis,

$\Omega_{ROTv}, \Omega_{ROTh}$ are angular velocities of the propelers,

$K_{v1}, \ldots, K_{v7}, K_{h1}, \ldots, K_{h7}$ are parameters of the controller,

$U_{v\max}, U_{h\max}$ are maximum values of the vertical and horizontal control.

The positions of the parameters are shown in the table below.

Table 5. Position of the parameter for the extended state-feedback controller.

Position	Parameter	Position	Parameter
1	K_{v1}	9	K_{h2}
2	K_{v2}	10	K_{h3}
3	K_{v3}	11	K_{h4}
4	K_{v4}	12	K_{h5}
5	K_{v5}	13	K_{h6}
6	K_{v6}	14	K_{h7}
7	K_{v7}	15	$U_{v\max}$
8	K_{h1}	16	$U_{h\max}$

For algorithm number 99:

depends on the structure of the external algorithm. See the External Interface Manual - 33-007-3M5 for details.

See: *GetP, SetPWHoriz, GetPWVert*

Note: There is a GUI tool which can be used to tune the parameters of the controllers. Execute the *hl_cpar* command to call this tool (see Figure 2-11).

Figure 2-11: The result of the *hl_cpar* command.

Example: Start PI controller for vertical DC drive.

```
hl_call( 'SetAlgNo', 0 );          % stop excitation
hl_call( 'SetP', [                 1.774   0.0 ...      % Kpvv and Kpvh
                  0.5    0.0 ...   % Kivv and Kivh
                  0.0    0.0 ...   % Kdvv and Kdvh
                  0.0    0.0 ...   % Kphv and Kphh
                  0.0    0.0  ...  % Kihv and Kihh
                  0    0.0...      % Kdhv and Kdhh
                  1.194    0.0 ... % Ivvsat and Ivhsat
                  0.0    0.0 ...   % Ihvsat and Ihhsat
                  0.5 0.0 ] );     % Uvmax and Uhmax

hl_call( 'SetAlgNo', 2 );
```

2.29 SetPWHoriz

Purpose: Set the parameters of the internal excitation source for horizontal position.

Synopsis: *hl_call('SetPWHoriz', 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 horizontal DC drive in the case of open-loop control mode (algorithm number 1),
- a source of reference value for the horizontal 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 2-12):

Par(2) is the level of the signal,

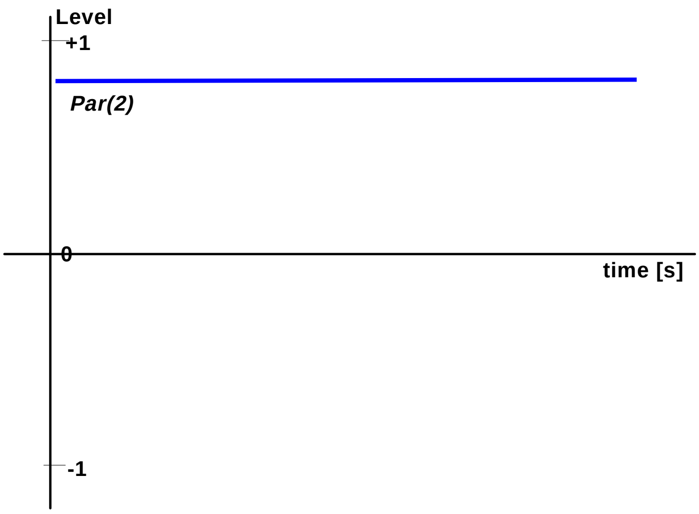


Figure 2-12: Constant excitation.

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

Par(2), Par(3), Par(4), Par(5) - time periods,

Par(6), Par(7), Par(7), Par(8) - appropriate levels,

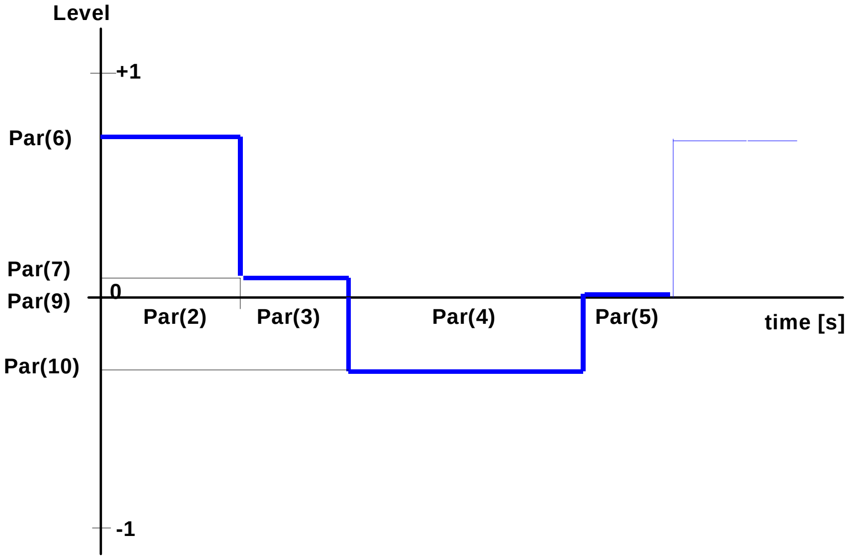


Figure 2-13: Square excitation.

Par(1) = 2: triangle wave. In this case (Figure 2-14):

Par(2), Par(3) - time periods,

Par(4), Par(5) - appropriate levels,

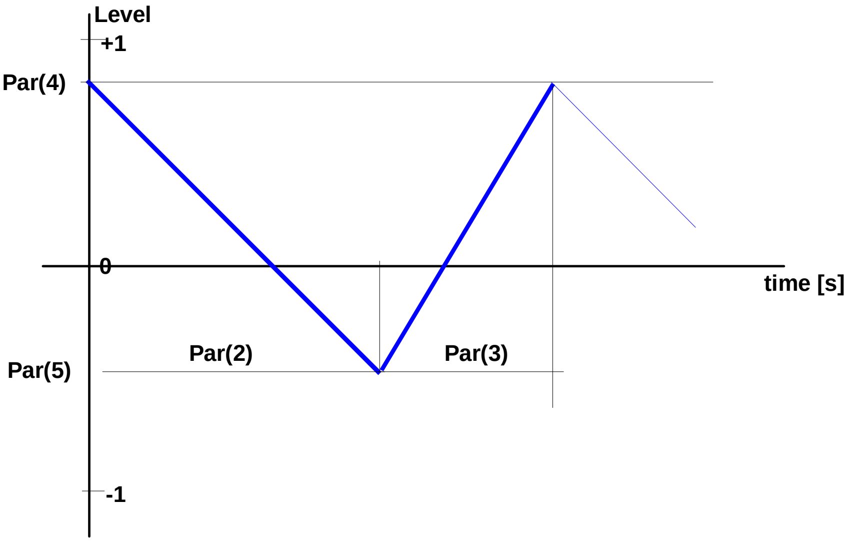


Figure 2-14: Triangle excitation.

Par(1) = 3: sinusoidal wave. In this case (Figure 2-15

Par(2) - time period,

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

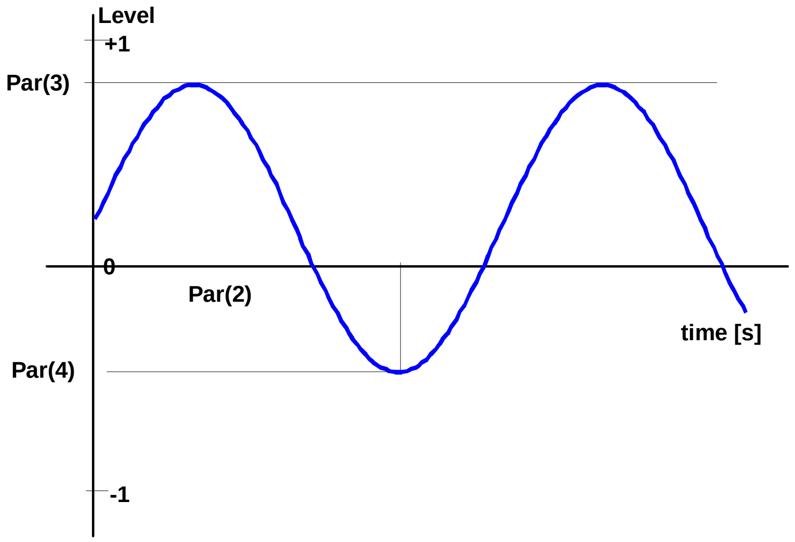


Figure 2-15: Sinusoidal excitation.

Par(1) = 4: random wave. In this case (Figure 2-16):

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.

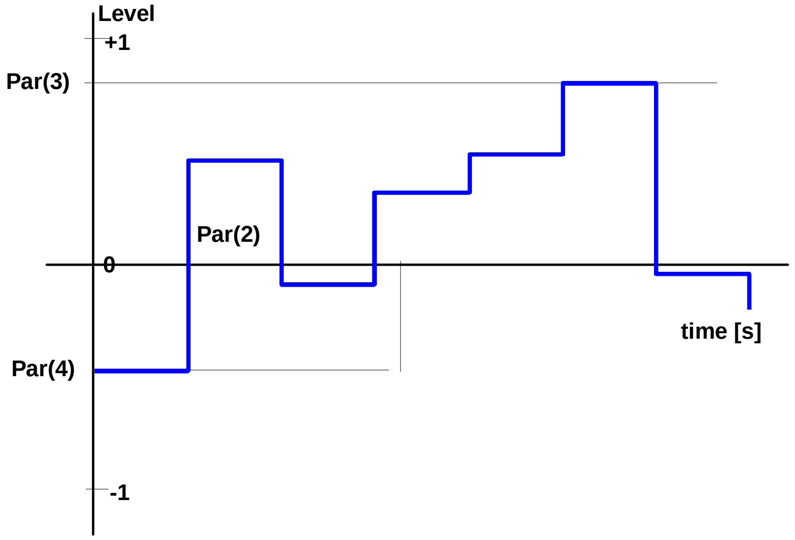


Figure 2-16: Random excitation.

Note: There is a GUI tool which can be used to tune the parameters of the open-loop excitation for the horizontal DC drive. Execute the *hl_heg* command to call this tool (see figure below).

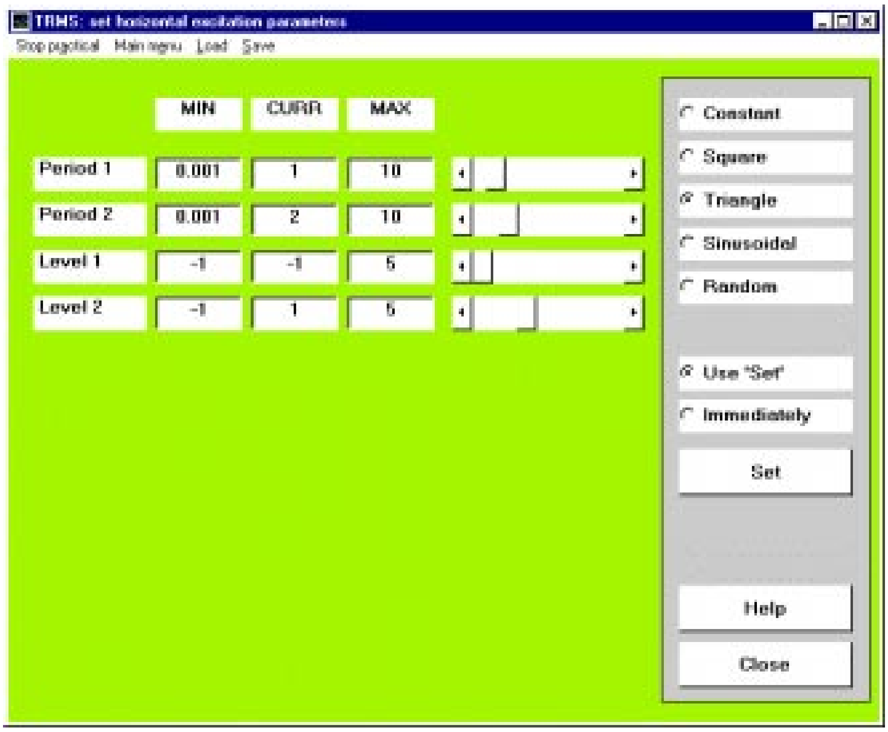


Figure 2-17: The *hl_heg* GUI interface.

See: *GetPWHoriz, 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. Start an open-loop experiment.

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

2.30 SetPWVert

Purpose: Set the parameters of the internal excitation source for vertical position.

Synopsis*:* *hl_call('SetPWVert', Par)*

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

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

See description of the *SetPWHoriz* function for details.

See*:* *GetPWVert, SetAlgNo, SetPWHoriz*

Note*:* There is a GUI tool which can be used to tune the parameters of the open-loop excitation for the vertical DC motor executed by the *hl_veg* command.

2.31 SetSampleTime

Purpose: Set basic clock.

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

Description: The function sets the basic clock of RTK. The sampling period of A/D converter is set by the 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 interval of the sampling period use the following statements:

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

2.32 SetVertPosFilter

Purpose: Set the parameters of the vertical position digital filter.

Synopsis*:* *ret = hl_call('SetVertPosFilter', fi)*

Description: The function sets the parameters of the digital filter currently active in RTK.

The filter associated with the *SetVertPosFilter* function operates on the vertical position data of the beam.

See description of the *SetHorizPosFilter* for filter details.

See*:* *GetVertPosFilter, SetVertRotorFilter, SetVertSpeedFilter*

2.33 SetVertRotorFilter

Purpose: Set the parameters of the digital filter for the velocity rotor velocity.

Synopsis: *ret = hl_call('GetVertRotorFilter', fi)*

Description: The function sets the parameters of the digital filter currently active in RTK.

The filter associated with the *SetVertRotorFilter* function operates on the velocity data of the tail rotor.

See description of the *SetHorizPosFilter* for filter details.

See: *GetVertRotorFilter, SetVertPosFilter, SetVertSpeedFilter*

2.34 SetVertSpeedFilter

Purpose: Set the parameters of the vertical velocity digital filter.

Synopsis*:* *ret = hl_call('SetVertSpeedFilter', fi)*

Description: The function sets the parameters of the digital filter currently active in RTK.

The filter associated with the *SetVertSpeedFilter* function operates on the vertical velocity data of the beam.

See description of the *SetHorizPosFilter* for filter details.

See*:* *GetVertSpeedFilter, SetVertPosFilter, SetVertRotorFilter*

2.35 StartAcq

Purpose: Clear content of the buffer.

Synopsis*:* *hl_call('StartAcq')*

Description: The function erases the contents of the buffer. The action is similar to that of the *GetHistory* function, but does not return the content of the buffer.

Note that the function *GetNoOfSamples* returns zero when invoked immediately after a *StartAcq* call**.**

See*:* *GetNoOfSamples*

Example*:* The call to the *GetNoOfSamples* function immediately after the call to the *StartAcq* function gives different results depending of sample period lengths and computer speed.

For instance,

```
hl_call( 'SetSampleTime', 0.1 )
hl_call( 'AtartAcq' ); hl_call( 'GetNoOfSamples' )
ans =
     0
hl_call( 'SetSampleTime', 0.01 )
hl_call( 'AtartAcq' ); hl_call( 'GetNoOfSamples' )
ans =
     3
```

2.36 StopPractical

Purpose: Stop practical.

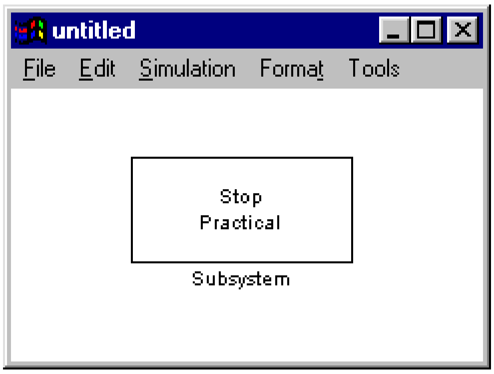
Synopsis*:* *hl_call('StopPractical')*

Description: The function is called to stop the practical. It suspends the currently active RTK control algorithm**.** The action is similar to *hl_call('SetAlgNo', 0)*.

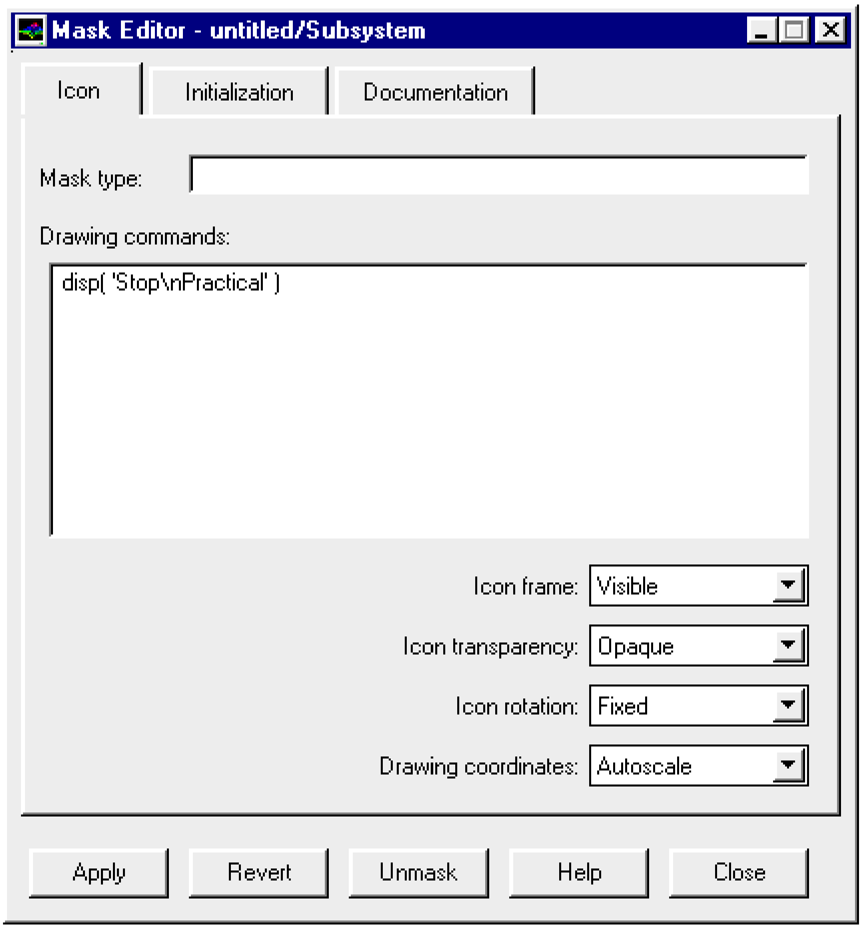
See*:* *SetAlgNo*

Example*:* You can create the Simulink model containing a single block. Double click on this block to execute the *hl_call('StopPractical')* function.

The Simulink model is shown below.



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



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

```
set_param( 'untitled/Subsystem', 'OpenFcn' , ...
           'hl_call(''StopPractical'');' )
```

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

2.37 UnloadLibrary

Purpose: Remove RTK library from memory.

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

Description: The command removes the library from the memory.

See*:* *LoadLibrary*

Note: The RTK is written in the form of a MATLAB mex-file. It causes any call to the *clear all* or *clear hl_call* to terminate the operation of the RTK and removes the library from the memory.

The termination of the MATLAB program removes the RTK from the memory automatically.

Notes

3. Information Flow Between Simulink Models and 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 construct a generic simulation block to handle, in one standard form, different roles, such as continuous simulation, discrete simulation, systems embedded within systems, and so on.

In our applications special S-functions perform information exchange between Simulink models and Real-Time Kernel. This section explains the usage of Simulink models as the graphical front-end of a real-time control program. 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 available to the MATLAB environment.

The masking mechanism, available in Simulink, allows you to define a block in terms of its dialogue box, its icon and initialisation commands.

Familiarity with the S-function format is assumed, and therefore 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.

3.1 Simulink models

This section contains a description of Simulink models designed for MATLAB version 5. The Simulink models and S-functions which co-operate witch the TRMS real-time task are also explained.

3.1.1 Guide for Simulink models and S-functions (example)

We shall discuss the S-function of the *PID Controller Tracking Mode* block. This block belongs to the main demo of the TRMS Simulink model called by the *hl* command in the MATLAB Command Window.

The Simulink model executed after a double click on this block is shown in Figure 3-1.

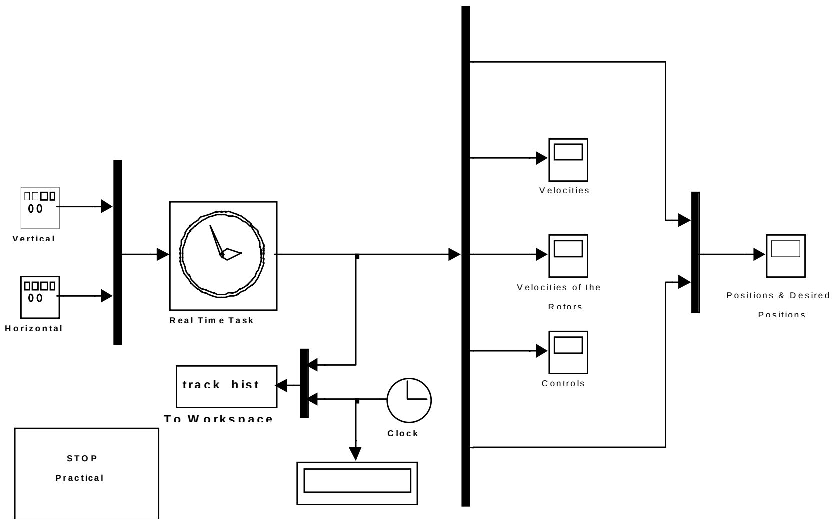


Figure 3-1: *PID Controller Tracking Mode* block.

The main block in the *PID Controller Tracking Mode* Simulink model is the *Real Time Task* block. It contains the S-function which performs data exchange between the RTK and the Simulink model.

The *Vertical* and *Horizontal* Simulink signal generators are sources of the reference position of the beam. The values of the reference positions are passed to the RTK by the S-function included in the *Real Time Task* block.

The second task of the *Real Time Task* block is to read from the RTK the current values of the vertical and horizontal position and vertical and horizontal velocity of the beam, values of the control for the DC drives and values of the desired position of the beam and send them to the output of the *Real Time Task* block. These values can be processed and visualised by any of the Simulink blocks.

For example, they can be shown in a graphical form in Simulink *Scope* blocks, in a numerical form in *Display* blocks or they can be transferred to the MATLAB workspace.

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

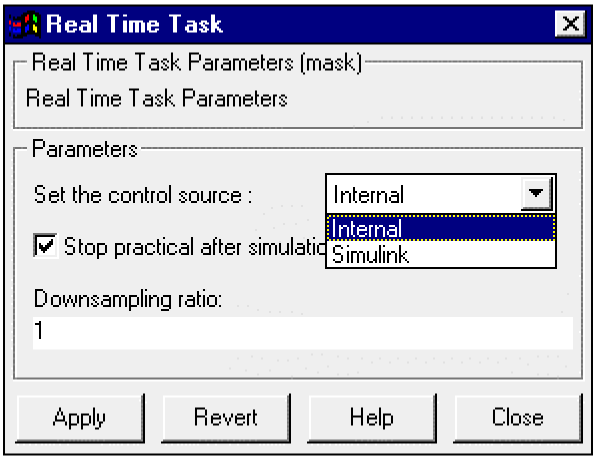


Figure 3-2: *Real-Time Task* dialogue box.

The window shown in Figure 3-2 sets 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 parameters are:

Set the control source - enables passing the new values of the desired positions from *Vertical* and *Horizontal* Simulink signal generators. There are two values available for this parameter:

Internal - the sources of the reference positions are the signal generators build into the RTK. They parameters are set in the S-function *hl_pirsf.m*,

Simulink - the sources of the reference positions are the Simulink signal generators *Vertical* and *Horizontal*,

Stop practical after simulation stop - flag used to stop real-time experiment after the Simulink simulation is stopped,

Downsampling ratio - defines the coefficient which decreases the amount of data produced by the *Real Time Task* block. For slow computers this parameter should be greater then one.

The *Real-Time Kernel* block is masked. To see the details of this block, execute the *Look under mask* command from the *Edit* menu.

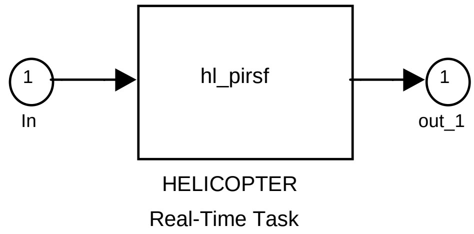


Figure 3-3: Unmasked Real-Time Kernel block.

The *Real-Time Kernel* block contains one main block: S-function labelled as *HELICOPTER Real-Time Task*. The blocks *In* and O*ut_1* from Fig.3.3 are required for masking. The name of the S-function connected with the S-function block is *hl_pirsf*. This S-function may be found in *hl_pirsf.m* file.

A double click on the S-function block opens the following window (Fig.3.4).

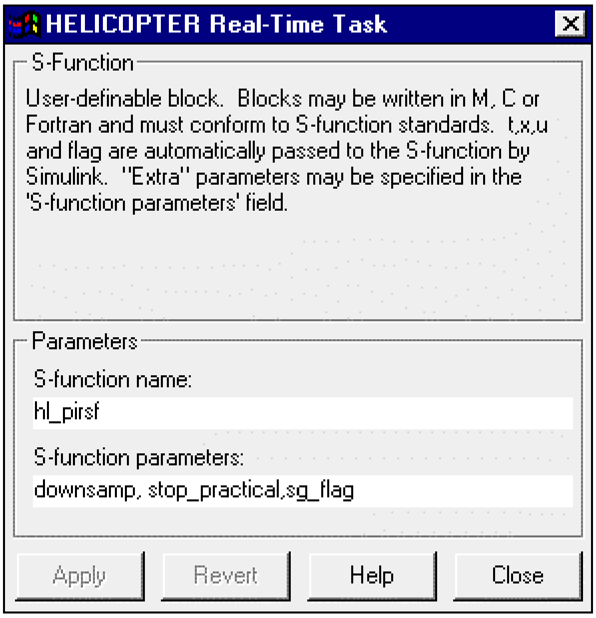


Figure 3-4: S-function dialogue box.

The user can set the subsystem S-function name and parameters in the window shown in Fig.3.4. In this example the function name is *hl_pirsf*. The names of the additional function parameters are set during masking of *Real-Time Kernel* block and are defined by *Edit Mask* command. See the *Initialisation* tab in Fig.3.5.

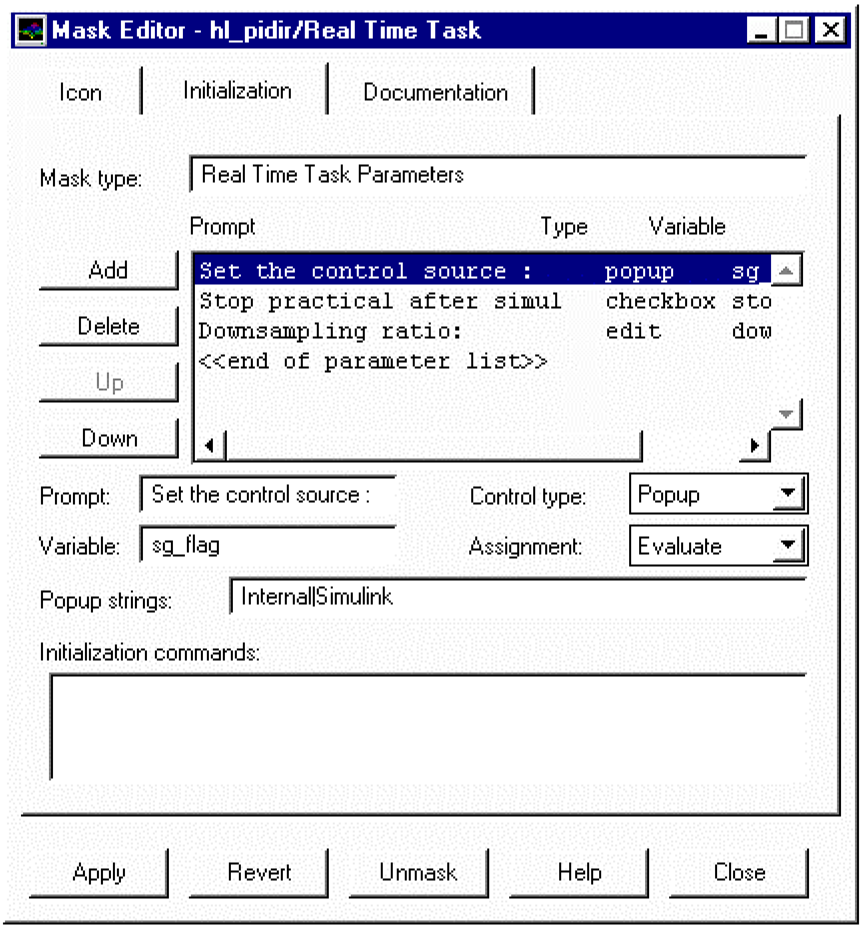


Figure 3-5: Mask definition for *Real-Time Kernel* block.

3.1.2 S-function - example 1

The content of the *hl_pirsf* S-function is described in the following part of this section. The lines written in italics are statements of the *hl_pirsf.m* file. The lines marked on the left side contain comments added for explaining statements of the S-function.

The *hl_pirsf* function sets the parameters of the PID controller embedded into the RTK and transfers the contents of the data acquisition buffer to Simulink. It contains the following parameters:

downsamp - decimation ratio of the output data stream,

stop_practical - the flag used to stop real-time experiment when simulation is stopped,

sg_flag - the flag which is used to enable changing the values of the desired positions.

The body of the *hl_pirsf* S-function is shown below.

function [sys, x0, str, ts] = sfunc(t, x, u, flag, downsamp, stop_practical, sg_flag)

The function has the following parameters:

t - time,

x - state vector,

u - input to the S-function block,

flag - the value passed to S-function by Simulink to distinguish different actions. The arguments *t*, *x*, *u* and *flag* are set and passed to S-function by Simulink automatically,

downsamp - downsampling ratio. Defines how many samples is 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 Real-Time Kernel to the output of the S-function block,

stop_practical - flag used to switch-off simulation. If *flag9* is set to 1 the control is off after executing *Simulation*/*Stop* command,

sg_flag - simulink generator flag. If this value is set to 2 Simulink generator is used as a source of desired position of the beam. The Simulink generator is connected to the input of the S-function block.

global hl_par history pos_in_history

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

VertPer = 20; % Vertical period [sec]

HorizPer = 10; % Horizontal period [sec]

VertLev = 0.2; % Vertical level

HorizLev = 1.0; % Horizontal level

Parameters of the internal source of the desired position.

switch flag,

case 0, % Initialization

```
    % Set number of continuous states, number of discrete states, number of
    % outputs and number of inputs.
    sizes.NumContStates  = 0;
    sizes.NumDiscStates  = 1;
    sizes.NumOutputs  = 10;
    sizes.NumInputs  = 2;
    sizes.DirFeedthrough  = 0;
    sizes.NumSampleTimes  = 1
    sys  = simsizes(sizes);
```

Set number of continuous states, number of discrete states, number of outputs and number of inputs (0 continuous states, 1 discrete state, 10 outputs, 2 inputs, 0 direct feedthrough - without algebraic loops, 1 sample time)

```
    dummy = hl_call( 'SetAlgNo', 0 );
    dummy = hl_call( 'SetP', [ 0.5 0 2.8 0 4.3 0 2.4 2.196 0 1.394 0 1.98 1.3 0 0 ...
```

```
                        1.87 1 1 ]);
dummy = hl_call( 'SetAlgNo', 2 );     % PID
```

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

```
dummy = hl_call( 'ResetTime', 0 );
pv = hl_call( 'GetPWVert' );
ph = hl_call( 'GetPWHoriz' );

% Set squere waves
pv( 1 : 9 ) = [ 1 VertPer/2  VertPer  VertPer  VertPer/2  -VertLev  0 ...
                VertLev  -VertLev ];
ph( 1 : 9 ) = [ 1 HorizPer/2 HorizPer HorizPer HorizPer/2 -HorizLev 0 ...
                HorizLev -HorizLev ];
dummy = hl_call( 'SetPWVert', pv );
dummy = hl_call( 'SetPWHoriz', ph );
```

Reset experiment time and set parameters of the desired position generator.

```
hl_par(1) = hl_call( 'GetSampleTime' );
hl_par(2) = hl_call( 'GetDivider'    );
```

Get sample period and sample period divider.

```
while ( hl_call( 'GetNoOfSamples' ) <= downsamp )
        ;
end;
history=hl_call( 'GetHistory' )';
```

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

```
pos_in_history = 1;

x0 = max( history( :, 1) );
```

Set initial conditions

```
str = [];    % str is always an empty matrix

% initialize the array of sample times
ts  = [-2 0];      % variable sample time


case 1, % Unhandled flags
    sys = [];
```

case 2, % Update - set new reference position

Set reference position if the *sg_flag* equal to 2.

```
if eq( sg_flag, 2 )
   pv = hl_call( 'GetPWVert' );
   ph = hl_call( 'GetPWHoriz' );
   pv( 1 : 2 ) = [ 0 u( 1 ) ];
   ph( 1 : 2 ) = [ 0 u( 2 ) ];
   dummy = hl_call( 'SetPWVert', pv );
   dummy = hl_call( 'SetPWHoriz', ph );
end

sys = x;
```

Return dummy value. This action is required by S-function.

case 3, % Outputs - return samples

Calculate output from the S-function block. The downsample coefficient *downsamp* is used to select the sample which is returned as the output from the S-function block. The *downsamp* variable is used to send only one sample of every *downsamp* samples from Real-Time Kernel to the output.

```
sys = history( downsamp, 2:11 );
curr_len = size( history, 1 );
history = history( downsamp + 1 : curr_len, : );
```

case 4, % GetTimeOfNextVarHit

Calculate next discrete time point. The S-function block will be activated in this time point. Waits for the appropriate number of samples in the data acquisition buffer to perform the time synchronisation between the Simulink model and the real time.

```
curr_len = size( history, 1 );
if( curr_len < downsamp )
 while ( hl_call( 'GetNoOfSamples' ) + curr_len <= downsamp )
   ;
 end;
 history = [ history; hl_call( 'GetHistory' )' ];
end
sys = history( downsamp, 1 );
```

The next discrete time point is taken from real-time data acquisition buffer. **This action synchronises simulation time of the Simulink model and real time of the Real-Time Kernel**.

case 9, % Terminate

Terminate simulation. The *stop_practical* variable is used to switch off the control value. This variable is set to 1 if the *Stop practical after simulation stop* check box is selected (see Figure 3-2)

```
if stop_practical == 1
 dummy = hl_call( 'StopPractical' );
```

```
    dummy = hl_call( 'SetSampleTime', 0.05 );
  end

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

end
```

3.1.3 S-function - example 2

The *hl_anima* Simulink model is based on the *PID Controler Tracking Mode* model (See Figure 3-1). This model is shown in Figure 3-6. Compared to the *PID Controler Tracking Mode* model, the *hl_anima* model contains one new block - the *TRMS Animation* block.

The S-function associated with the *TRMS Animation* block demonstrates how to create a simple animation in the MATLAB environment.

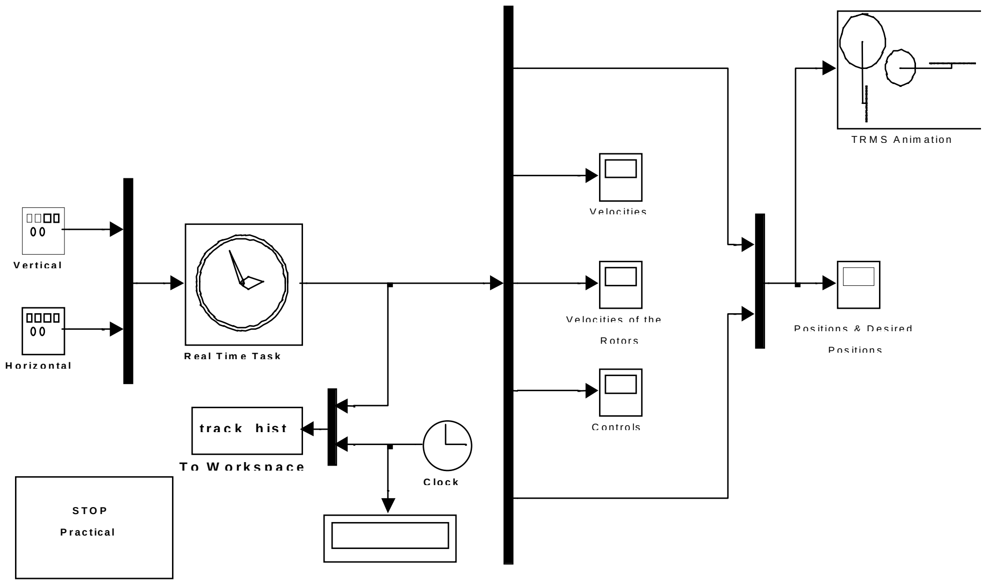


Figure 3-6: The *hl_anima* Simulink model.

The *hl_anima* model is a part of the TRMS toolbox and is stored in the *hl_anima.mdl* file.

When you double click on the *TRMS Animation* block the following window appears (Figure 3-7).

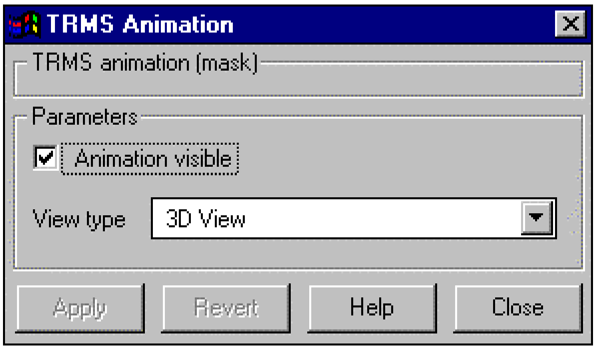


Figure 3-7: Parameters of the *TRMS Animation* block.

The *TRMS Animation* block operates in one of two modes and displays the graphical view of the real model. The examples of different modes are shown in Figure 3-8 and Figure 3-9. The first presents the 3D view of the current position of the model and the reference position of the beam. The second displays two views of the positions of the vertical and horizontal axis and two reference positions.

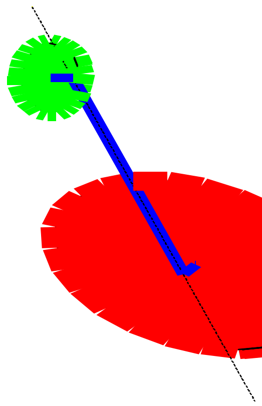


Figure 3-8: 3D view of the TRMS model.

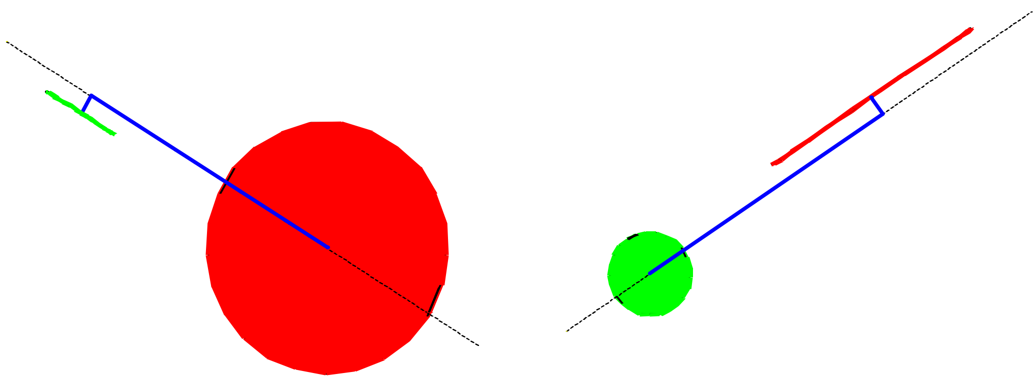


Figure 3-9: 2D view of the TRMS model.

The *TRMS Animation* block is masked. Details of the unmasked block are given in Figure 3-10.

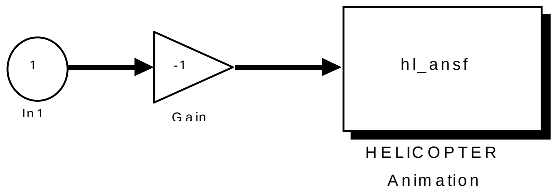


Figure 3-10: Unmasked *TRMS Animation* block.

One can see that this block executes the S-function included in *hl_ansf.m* file.

When you mask the *TRMS Animation* block the following two windows appear (Figure 3-11).

The first windows defines the view of the icon displayed in the *TRMS Animation* block.

The second defines two parameters:

Animation visible - the flag which enables to display the animation window,

View type - the parameter used to change the view type.

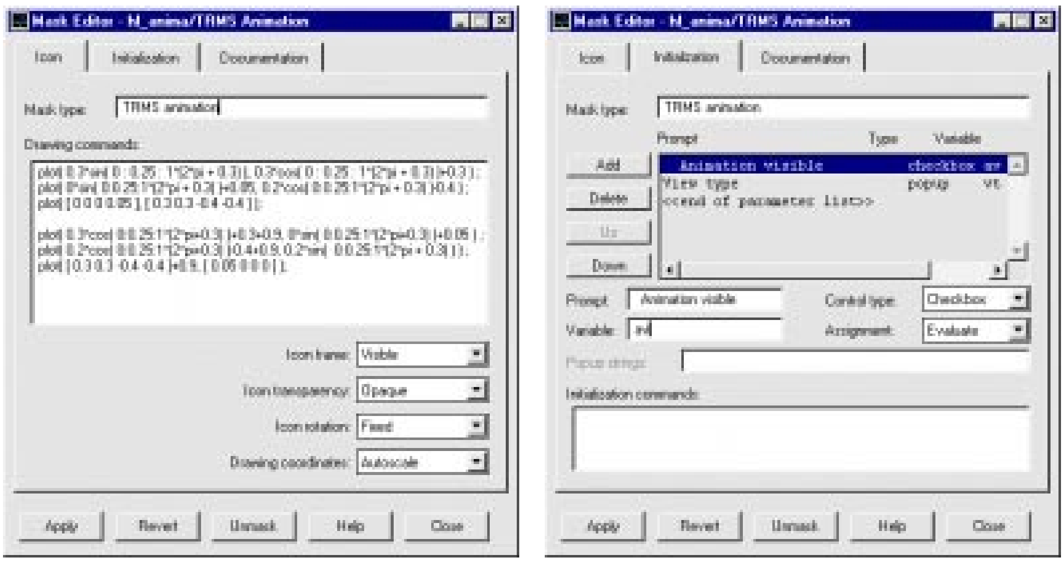


Figure 3-11: Masking *TRMS Animation* block.

The body of the *hl_ansf* S-function is shown below.

function [sys, x0, str, ts] = hl_ansf(t, x, u, flag, av, vt)

This function creates the window and animates position of the TRMS model.

The function has the following parameters:

t	- time,
x	- state vector,
u	- input to the S-function block. It contains four values: vertical and horizontal position of the beam and two reference positions,
flag	- the value passed to S-function by Simulink to distinguish different actions. The arguments *t*, *x*, *u* and *flag* are set and passed to S-function by Simulink automatically,
av	- the flag which enable the animation window,
vt	- the flag which selects the 3D or 2D view.

Set title of the animation window.

fig_name = 'Animation of the helicopter model';

Set auxiliary variables used to display the graphical view of the animation window.

ForeLen = 0.3;

BackLen = 0.4;

BaseHeigth = 0.0;

VertRotRad = 0.3;

HorizRotRad = 0.1;

RotorDist   = 0.05;

```
ax = 0.7 * [ -1 1 -1 1 -1 1 ];

% Data to plot the helicopter model
v  = 0 : 0.25 : 2*pi + 0.3;

% Line
lx  = [ 0 0 0 -RotorDist ];
ly  = [ ForeLen ForeLen -BackLen -BackLen ];
lz  = [ RotorDist 0 0 0 ];

% Vertical rotor
vrx = VertRotRad * sin( v );
vry = VertRotRad * cos( v ) + ForeLen;
vrz = 0 * v + RotorDist;

% Horizontal rotor
hrx = 0 * v - RotorDist;
hry = HorizRotRad * sin( v ) - BackLen;
hrz = HorizRotRad * cos( v );

% Line - desired position
dlx  = [    0      0 ];
dly  = [ -0.65  0.75 ];
dlz  = [    0      0 ];

switch flag,

case 0,    % Initialization
```

Check the visibility flag (*av* variable) and the view type (*vt* variable)

```
% av: 0 - hidden,  1 - visible
% vt: 1 - 3D view, 2 - 2D view
% Check, if the anomation window was created previously
if av == 1
  if vt == 1
```

3D view selected.

Test presence of the animation window

```
    h_arr = get( 0, 'Children' );
    position=get(0, 'DefaultFigurePosition' );
    h = -99;

    % Find the handler to the animation window
    for i = 1 : length( h_arr )
      if strcmp(get( h_arr(i), 'Type'), 'figure')
        if strcmp( get( h_arr( i ), 'Name' ), fig_name )
          h = h_arr( i );
          delete( get(h_arr( i ), 'Children' ) );
          break;
        end;
      end;
    end;
```

Create a new window if animation window is absent.

```
    % Create new window
    if ( h < 0 )
```

```
  h = figure( 'Name',  fig_name, 'NumberTitle', 'off', ...
           'Position', position, 'Color', [ 0 0 0 ] );
end;
```

Set ranges of the horizontal and vertical axis.

```
axes( 'Xlim', ax(1:2),  'Ylim', ax(3:4), 'Zlim', ax(5:6), ...
   'Visible', 'off', 'View', [ 90 0 ] );
hold on;
```

Plot the helicopter model. Set visible mode to off

```
e1 = plot3( lx, ly, lz,    'b', 'Visible', 'off', 'EraseMode', 'xor' );
e2 = plot3( vrx, vry, vrz, 'r', 'Visible', 'off', 'EraseMode', 'xor' );
e3 = plot3( hrx, hry, hrz, 'g', 'Visible', 'off', 'EraseMode', 'xor' );
e4 = patch( vrx, vry, vrz, 'r', 'Visible', 'off', 'EraseMode', 'xor' );
e5 = patch( hrx, hry, hrz, 'g', 'Visible', 'off', 'EraseMode', 'xor' );
e6 = plot3( dlx, dly, dlz, 'y', 'Visible', 'off', 'EraseMode', 'xor' );
```

Set line properties

```
set( e1, 'LineWidth', 8 );
set( e2, 'LineWidth', 18 );
set( e3, 'LineWidth', 18 );
set( e6, 'LineWidth', 1, 'LineStyle', '--' );
```

Set UserData properties. They are used to store "static" data.

```
  set( h, 'UserData', [ e1 e2 e3 e4 e5 e6 ] );
```

Plot 2D view if the *vt* flag is equal to 2. The statements are similar as in the case of the 3D view.

```
elseif vt == 2
  % Check, if the anomation window was created previously
  h_arr = get( 0, 'Children' );
  position=get(0, 'DefaultFigurePosition' );
  position( 3 ) = 1.20 * position( 3 );
  position( 4 ) = 0.75 * position( 4 );
  h = -99;

  % Find the handler to the animation window
  for i = 1 : length( h_arr )
    if strcmp(get( h_arr(i), 'Type'), 'figure')
      if strcmp( get( h_arr( i ), 'Name' ), fig_name )
        h = h_arr( i );
        delete( get(h_arr( i ), 'Children' ) );
        break;
      end;
    end;
  end;

  % Create a new window
  if ( h < 0 )
```

```
  h = figure( 'Name',  fig_name, 'NumberTitle', 'off', ...
            'Position', position, 'Units', 'Normalized', ...
            'Color', [ 0 0 0 ] );
end;

% Set ranges of the horizontal and vertical axis
hl_Ax1 = axes( 'Xlim', ax(1:2),  'Ylim', ax(3:4), 'Zlim', ax(5:6), ...
          'Visible', 'off', 'View', [ 0 -90 ], ...
          'Position', [ 0 0 0.5 1 ], 'NextPlot', 'Add' );
% Plot the helicopter model - visible mode set to off
e1 = plot3( lx, ly, lz,    'Visible', 'off', 'EraseMode', 'xor' );
e2 = plot3( vrx, vry, vrz, 'Visible', 'off', 'EraseMode', 'xor' );
e3 = plot3( hrx, hry, hrz, 'Visible', 'off', 'EraseMode', 'xor' );
e4 = patch( vrx, vry, vrz, 'r', 'Visible', 'off', 'EraseMode', 'xor' );
e5 = patch( hrx, hry, hrz, 'g', 'Visible', 'off', 'EraseMode', 'xor' );
e6 = plot3( dlx, dly, dlz,  'Visible', 'off', 'EraseMode', 'xor' );

hl_Ax2 = axes( 'Xlim', ax(1:2),  'Ylim', ax(3:4), 'Zlim', ax(5:6), ...
          'Visible', 'off', 'View', [ 90 0 ], ...
          'Position', [ 0.5 0 0.5 1 ], 'NextPlot', 'Add' );
% Plot the helicopter model - visible mode set to off
e11 = plot3( lx, ly, lz,    'Visible', 'off', 'EraseMode', 'xor' );
e12 = plot3( vrx, vry, vrz, 'Visible', 'off', 'EraseMode', 'xor' );
e13 = plot3( hrx, hry, hrz, 'Visible', 'off', 'EraseMode', 'xor' );
e14 = patch( vrx, vry, vrz, 'r', 'Visible', 'off', 'EraseMode', 'xor' );
e15 = patch( hrx, hry, hrz, 'g', 'Visible', 'off', 'EraseMode', 'xor' );
e16 = plot3( dlx, dly, dlz,  'Visible', 'off', 'EraseMode', 'xor' );

set( e1, 'Color', 'b', 'LineWidth', 2 );
set( e2, 'Color', 'r', 'LineWidth', 4 );
set( e3, 'Color', 'g', 'LineWidth', 4 );
```

```
    set( e6, 'Color', 'y', 'LineWidth', 1, 'LineStyle', '--' );
    set( e11, 'Color', 'b', 'LineWidth', 2 );
    set( e12, 'Color', 'r', 'LineWidth', 4 );
    set( e13, 'Color', 'g', 'LineWidth', 4 );
    set( e16, 'Color', 'y', 'LineWidth', 1, 'LineStyle', '--' );

    % Set UserData properties
    set( h, 'UserData', [ e1 e2 e3 e4 e5 e6 e11 e12 e13 e14 e15 e16 hl_Ax1 hl_Ax2 ] );
  end
end
```

> Initialise sizes - 0 continuous states, 0 discrete state, 0 outputs, 4 inputs

```
sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 0;
sizes.NumInputs      = 4;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;     % at least one sample time is needed

sys = simsizes(sizes);

x0  = [ ];
str = [ ];    % str is always an empty matrix
ts  = [ ];
```

```
case 2,    % Update state
```

Find the handler of the animation window.

```
h_arr = get( 0, 'Children' );
h = -99;

% Find handle to the animation window
for i = 1 : length( h_arr )
  if strcmp(get( h_arr(i), 'Type'), 'figure')
    if strcmp( get( h_arr( i ), 'Name' ), fig_name )
      h = h_arr( i );
      break;
    end;
  end;
end;
```

Plot if the animation window exists.

```
if ( h > 0 )   % If animation window exists
```

Select the appropriate view type.

```
  if av == 1
    if vt == 1
```

3D view selected

Set current position of the beam and reference position.

```
fi  = u( 1 );   psi  = u( 2 );
dfi = u( 3 );   dpsi = u( 4 );
```

Obtain "static" *UserData*. They contain the handlers to the graphical objects.

```
aux = get( h, 'UserData' );
e1 = aux( 1 ); e2 = aux( 2 ); e3 = aux( 3 );
e4 = aux( 4 ); e5 = aux( 5 ); e6 = aux( 6 );
```

Calculate the current positions of the graphical objects.

```
[ xppe1, yppe1, zppe1 ]    = hl_anitr( lx,  ly,  lz,  fi, psi, 0 );
[ xppe24, yppe24, zppe24 ] = hl_anitr( vrx, vry, vrz, fi, psi, 0 );
[ xppe35, yppe35, zppe35 ] = hl_anitr( hrx, hry, hrz, fi, psi, 0 );
[ xdlx, ydly, zdlz ]       = hl_anitr( dlx, dly, dlz, dfi, dpsi, 0 );
```

Set and display the new positions of the graphical objects.

```
  set( e1, 'XData', xppe1,  'YData', yppe1,  'ZData', zppe1 );
  set( e2, 'XData', xppe24, 'YData', yppe24, 'ZData', zppe24 );
  set( e3, 'XData', xppe35, 'YData', yppe35, 'ZData', zppe35 );
  set( e4, 'XData', xppe24, 'YData', yppe24, 'ZData', zppe24 );
  set( e5, 'XData', xppe35, 'YData', yppe35, 'ZData', zppe35 );
  set( e6, 'XData', xdlx,   'YData', ydly,   'ZData', zdlz );

  set( e1, 'Visible', 'on' );  set( e2, 'Visible', 'on' );
  set( e3, 'Visible', 'on' );  set( e4, 'Visible', 'on' );
  set( e5, 'Visible', 'on' );  set( e6, 'Visible', 'on' );
```

If *vt* variable is equal to 2 plot the 2D view. The statements are similar to that for the 3D view.

```
elseif vt == 2
  fi  = u( 2 )+pi/2;   psi  = u( 1 );
  dfi = u( 4 )+pi/2;   dpsi = u( 3 );

  aux = get( h, 'UserData' );
  e1 = aux( 1 ); e2 = aux( 2 ); e3 = aux( 3 );
  e4 = aux( 4 ); e5 = aux( 5 ); e6 = aux( 6 );
  e11 = aux( 7 ); e12 = aux( 8 ); e13 = aux( 9 );
  e14 = aux( 10 ); e15 = aux( 11 ); e16 = aux( 12 );
  hl_Ax1 = aux( 13 );  hl_Ax2 = aux( 14 );

  %
  % Azimuth
  %
  [ xp, yp, zp ]    = hl_anitr( lx,  ly,  lz,  0, fi, 0 );
  set( e1, 'XData', xp, 'YData', yp, 'ZData', zp );
```

```
[ xp, yp, zp ]    = hl_anitr( hrx,  hry,  hrz,  0, fi, 0 );
set( e3, 'XData', xp, 'YData', yp, 'ZData', zp );
set( e5, 'XData', xp, 'YData', yp, 'ZData', zp );

[ xp, yp, zp ]    = hl_anitr( vrx,  vry,  vrz,  0, fi, 0 );
set( e2, 'XData', xp, 'YData', yp, 'ZData', zp );
set( e4, 'XData', xp, 'YData', yp, 'ZData', zp );

[ xp, yp, zp ]    = hl_anitr( dlx,  dly,  dlz,  0, fi, 0 );
set( e6, 'XData', xp, 'YData', yp, 'ZData', zp );

%
% Elevation
%
[ xp, yp, zp ]    = hl_anitr( lx,  ly,  lz,  psi, 0, 0 );
set( e11, 'XData', xp, 'YData', yp, 'ZData', zp );

[ xp, yp, zp ]    = hl_anitr( hrx,  hry,  hrz,  psi, 0, 0 );
set( e13, 'XData', xp, 'YData', yp, 'ZData', zp );
set( e15, 'XData', xp, 'YData', yp, 'ZData', zp );

[ xp, yp, zp ]    = hl_anitr( vrx,  vry,  vrz,  psi, 0, 0 );
set( e12, 'XData', xp, 'YData', yp, 'ZData', zp );
set( e14, 'XData', xp, 'YData', yp, 'ZData', zp );

[ xp, yp, zp ]    = hl_anitr( dlx,  dly,  dlz,  psi, 0, 0 );
set( e16, 'XData', xp, 'YData', yp, 'ZData', zp );


set( e1,  'Visible', 'on' );  set( e2,  'Visible', 'on' );
set( e3,  'Visible', 'on' );  set( e4,  'Visible', 'on' );
set( e5,  'Visible', 'on' );  set( e6,  'Visible', 'on' );
```

```
        set( e11, 'Visible', 'on' );  set( e12, 'Visible', 'on' );
        set( e13, 'Visible', 'on' );  set( e14, 'Visible', 'on' );
        set( e15, 'Visible', 'on' );  set( e16, 'Visible', 'on' );
      end
    end

  end
 sys = [ ];



case 1,    % Unhandled flag
  sys = [];
case 3,    % Unhandled flag
  sys = [ ];
case 4,    % Unhandled flag
  sys = [];
case 9,    % Unhandled flag
  sys = [];

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

end
```

The *hl_anitr* local function calculates the position of the animated model in the 3-dimensional space. It uses the following formulas:

*x' = x*cos(fi) + y*sin(fi)*

*y' = -x*sin(fi) + y*cos(fi)*

z' = z

x'' = x'
*y'' = y'*cos(psi) + z'*sin(psi)*
*z'' = -y'*sin(psi) + z'*cos(psi)*

where *fi* is angle around the OZ axis and *psi* is angle around the OX axis.

```
function [ xp, yp, zp ] = hl_anitr( x, y, z, fi, psi, theta )

x1 = x;
y1 =  y*cos( fi ) + z*sin( fi );
z1 = -y*sin( fi ) + z*cos( fi );

x2 =  x1*cos( psi ) + y1*sin( psi );
y2 = -x1*sin( psi ) + y1*cos( psi );
z2 = z1;

xp = x2;
yp = y2;
zp = z2;
```

Notes

4. Quick Reference Table

Function Name	Description
1. RTK communication	
GetAlgNo	number of algorithm
GetBaseAddress	get base address of PCL-812 board
GetDivider	auxiliary clock multiplier
GetHistory	content of the buffer
GetHorizPosFilter	set parameters of the horizontal position filter
GetHorizRotorFilter	set parameters of the horizontal rotor velocity filter
GetHorizSpeedFilter	set parameters of the horizontal velocity filter
GetModelP	get parameters of the built-in mathematical model
GetNrOfSamples	number of samples in the buffer
GetP	parameters of the control algorithm
GetPendPosFilt	get parameters of the pole position filter
GetPendSpeedFilt	get parameters of the pole velocity filter
GetPWHoriz	parameters of internal excitation generator - horizontal axis
GetPWVert	parameters of internal excitation generator - vertical axis
GetSampleTime	basic clock period
GetVertPosFilter	set parameters of the vertical position filter
GetVertRotorFilter	set parameters of the vertical rotor velocity filter
GetVertSpeedFilter	set parameters of the vertical velocity filter
LoadLibrary	load RTK DLL library
ResetEncoders	reset incremental encoders
ResetTime	set experiment's time to zero
SetAlgNo	selects the control algorithm and starts the experiment
SetBaseAddress	set base address of PCL-812 board

SetDivider	set auxiliary clock
SetHorizPosFilter	set parameters of the horizontal position filter
SetHorizRotorFilter	set parameters of the horizontal rotor velocity filter
SetHorizSpeedFilter	set parameters of the horizontal velocity filter
SetInitCond	set initial conditions for the built-in mathematical model
SetModelP	set parameters of the built-in mathematical model
SetP	set parameters of the controller
SetPWHoriz	parameters of internal excitation generator - horizontal axis
SetPWVert	parameters of internal excitation generator - vertical axis
SetSampleTime	set basic clock
SetVertPosFilter	set parameters of the vertical position filter
SetVertRotorFilter	set parameters of the vertical rotor velocity filter
SetVertSpeedFilter	set parameters of the vertical velocity filter
StartAcq	clear content of the buffer
StopPractical	stop practical
UnloadLibrary	remove RTK DLL library from memory

2. Demo m-files	
hl_cpar	select active controller; set parameters of active controller (GUI interface)
hl_heg	set parameters of the internal excitation - horizontal axis (GUI interface)
hl_mouse	use mouse to control or to set reference position (GUI interface)
hl_opens	S-function for open-loop control
hl_pids	S-function for stabilising PID controller
hl_pirsf	S-function for tracking PID controller
hl_saddr	set base address of the I/O board (GUI interface)
hl_sf	S-function for state feedback controller
hl_stet	plot contents of the data acquisition buffer (GUI interface)
hl_veg	set parameters of the internal excitation - vertical axis (GUI interface)

3. Demo Simulink models (mdl-files)	
hl	start main demo
hl_1doff	identification: tuning of 1DOF vertical subsystem
hl_anima	RTMS model animation
hl_omh	identification: tuning of time constant of tail rotor
hl_omv	identification: tuning of time constant of main rotor
hl_open	open-loop control
hl_pid	PID stabilising controller
hl_pidir	PID tracking mode
m_1doff	tuning of 1-DOF vertcal part of the TRMS model
m_1dofh	model of the horizontal part of the TRMS model
m_1dofv	model of the vertical part of the TRMS model
m_omh	identification of the time constants of the horizontal tail rotor
m_omv	identification of the time constants of the horizontal main rotor
ss_1dofh	simulation of the PID control for the 1DOF horizontal movement
ss_1dofv	simulation of the PID control for the 1DOF vertical movement
ss_2dofc	simulation of the cross-coupled PID control for the 2DOF system
ss_2dofs	simulation of the simple PID control for the 2DOF system