Twin Rotor MIMO System

External Interface

33-007-3M5

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: 33-007-3M5 Ed01 980930 Printed in England by FL Ltd, Crowborough

Feedback Part No. 1160-330073M5
TWIN ROTOR MIMO SYSTEM
External Interface

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](image1)

Refer to accompanying documents

![CAUTION - RISK OF ELECTRIC SHOCK](image2)

![CAUTION - ELECTROSTATIC SENSITIVE DEVICE](image3)

---

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.
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.

COPYRIGHT NOTICE

© Feedback Instruments Limited

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of Feedback Instruments Limited.

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 1-1

2. Structure of the Real-Time Kernel Dynamic Linked Library 2-1

3. External DLL Library 3-1

   3.1 Code and data segments limitations 3-5

   3.2 Creating 32-bit DLL libraries 3-5

   3.3 Borland C++ ver. 5.0 3-6
      3.3.1 WATCOM C/C++ v.10.6 3-7
      3.3.2 Microsoft Visual C++ v. 2.0 3-8

4. Exported Functions 4-1

   4.1 ExternalController function 4-1

   4.2 ExternalIPC function 4-5

5. Communication with Matlab. Reference List 5-1

   5.1 Function LoadExtAlg 5-2

   5.2 Function UnLoadExtAlg 5-5

   5.3 Function GetExtAlgName 5-6

   5.4 Function CallExtIPC 5-7

   5.5 Function ExtAlgOneStep 5-9

6. Example 1: Non-Linear Stabilising Controller with Tuned Coefficients 6-1

7. Example 2: Off-Line Testing of Externalcontroller 7-1

8. Example 3: Pid Controllers Stabilising Velocity of the Propellers 8-1

9. Example 4: Simulink Model for the External Controller 9-1

10. Quick Reference Table 10-1
Notes
1. Introduction

Although the Real-Time Kernel (RTK) for TRMS and the 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 (EI) to Real-Time Kernel was developed to extend features of the control software. The interface forms the way in which user-designed controllers can be added to the RTK and implemented in real time. To more experienced users the External Interface offers full access to the RTK, making the control technology more open.

The main part of this manual describes procedures for creating C-code DLL libraries. It should be noted that it is possible to develop similar procedures for Pascal, FORTRAN or other high level languages.

It is assumed that the reader is familiar with the following topics:

- RTK concept and function library for the TRMS system,
- MATLAB and Simulink environment,
- C programming language,
- basic MS-Windows API (Application Programming Interface) functions,
- Dynamic Linked Libraries concept.

It is also assumed that the reader is a licensed user of a C language compiler suitable to produce DLL executable files for Windows95/Windows NT operating system, the compiler is properly installed, and one knows how to use it.

The manuals begins with a short description of the Real-Time Kernel, presenting both its general structure and the principles of data exchange between modules of the system. It continues with a discussion of a structure of a DLL external library which is appropriate for creation of external controllers, followed by instructions on how to compile and link DLL libraries.

Section 4 describes the shells of two basic functions exported from the external DLL: the ExternalController and ExternalIPC functions for the TRMS model are described in detail.

In section 5 the reference list of function associated with the external DLL library is given.

Section 6 contains a short overview of the external controller.

Section 7 discusses the off-line testing of the external controller.
In section 8 the controller which stabilises the velocity of the propellers is described. Finally, the Simulink model and the S-function for the controller described in this section are discussed.

*Note: Example files of an external DLL library are delivered on the installation diskette or CDROM in the \EXTERNAL directory.*
2. Structure of the Real-Time Kernel Dynamic Linked Library

The general structure of the control system is shown in Figure 2-1. The main part is the RTK DLL library (the hl_call.dll file). It contains measurement procedures, digital filters, data acquisition buffer, built-in control algorithms, software to control actuators, internal excitation generator, set of controller parameters, MATLAB-to-RTK interface software and interface for external DLL libraries (see Figure 2-3). The RTK controls flow of all signals from and to the process. It contains functions performing analogue-to-digital and digital-to-analogue conversions.

The RTK DLL library is excited by timer interrupts. The main part of the RTK is executed during interrupt time. The Interface Software performs data format conversion between the RTK library and the MATLAB environment. The Inter-Process Communication Protocol is used to set parameters of the RTK and to read information from the RTK DLL library. Typically the RTK library is monitored by a MATLAB program, but any other Windows 95 or Windows NT program can be used in the supervisor layer. The RTK version for Windows NT operating system requires a special device driver (RTKIO.SYS) to enable the access to the I/O address space. The access to the I/O space is limited in Windows NT and that is why the device driver is absolutely necessary. All read and write operations for the I/O Board are performed using this driver (see Figure 2-2).

The RTK DLL library and the Interface Software are prepared to support External DLL libraries. The external library may contain a controller and interface software written in the special form required by the RTK DLL library. In the opposition to the built-in RTK algorithms the usage of the external DLL executable creates a possibility to add new control algorithms by the user. This solution extends features of the control software.

The external DLL library interface opens the way to add new controllers to the Real Time Kernel. The main advantage of the external interface as described, is that the user does not need to know the implementation details of the real-time program.

The basic knowledge required to implement this technique is described in this section.
CHAPTER 2

TWIN ROTOR MIMO SYSTEM
External Interface Structure of the Real-Time Kernel Dynamic Linked Library

Figure 2-1: General structure of the control system for MATLAB 5 and Windows 95.

Figure 2-2: General structure of the control system for MATLAB 5 and Windows NT.
The RTK DLL library (Figure 2-3) contains a set of built-in control algorithms selected by their numbers. To start experiments the RTK library must be loaded to the memory first. Next, the external DLL library is loaded. **The external DLL contains two main parts: the controller and the interface software.** To call the controller from the external DLL the **algorithm number 99** must be selected (Figure 2-3). In this case the RTK library calls the external library controller procedure during interrupt time. The RTK library passes the array of RTK parameters and all measurements to the controller procedure as arguments. The controller procedure returns control value for the DC drives.

![Diagram](image-url)  
*Figure 2-3: Block diagram of RTK.*
The external DLL library can be removed from the memory when the experiment is finished. Another external DLL library can be loaded and executed. However, it is not needed to unload the RTK library from the memory when changing the executed external DLL. The external DLL operating mode is a fast way to test different versions of control algorithms.
3. External DLL library

There is a number of issues that you need to understand in order to create your own external DLL libraries. This section describes the structure of such a library appropriate for development of user-defined controllers.

The components required to create the 32-bit DLL are shown in Figure 3-1.

Figure 3-1: Procedure for creating the 32-bit external DLL library

The C-source of the 32-bit external DLL library must contain the following functions:

- **DllMain** - function called by Windows95 or Windows NT when the DLL library is:
  - loaded for the first time by a given process,
  - DLL is unloaded by a given process,
  - a thread is created in a process that has already loaded this DLL,
  - a thread is exiting cleanly in a process that has already loaded this DLL.

The two last cases are not used by the RTK.

Use this function to initialise and perform a cleanup for an external controller. For example, the following code creates a simple *DllMain* function:

```c
#include <windows.h>
#include <stdio.h>
```
BOOL WINAPI DllMain( HINSTANCE hDLLInst, DWORD fdwReason, LPVOID lpvReserved )
{
    char msg_str[ 150 ];
    static char ModuleName[ 128 ];

    switch (fdwReason)
    {
    case DLL_PROCESS_ATTACH:
        // The DLL is being loaded for the first time by a given
        // process. Perform per-process initialization here. If
        // the initialization is successful, return TRUE; if
        // unsuccessful, return FALSE.
        GetModuleFileName( hDLLInst, ModuleName,
                           sizeof( ModuleName ) - 30 );
        sprintf( msg_str, "External DLL Library: %s",
                 ModuleName );
        MessageBox( (HWND)NULL,
                    "Entry point to external DLL reached",
                    msg_str, MB_OK );
        /*                                       */
        /* Insert your initialization code here */
        /*                                       */
        break;

    case DLL_PROCESS_DETACH:
        // The DLL is being unloaded by a given process. Do any
        // per-process clean up here, such as undoing what was
        // done in DLL_PROCESS_ATTACH. The return value is
        // ignored.
        sprintf( msg_str, "External DLL Library: %s",
                 ModuleName );
        MessageBox( (HWND)NULL,
                    "Exit point of external DLL reached",
                    msg_str, MB_OK );
    }
case DLL_THREAD_ATTACH:
    // A thread is being created in a process that has
    // already loaded this DLL. Perform any per-thread
    // initialization here. The return value is ignored.
    //
    // NOT USED in external DLLs
    break;
}

case DLL_THREAD_DETACH:
    // A thread is exiting cleanly in a process that has
    // already loaded this DLL. Perform any per-thread clean
    // up here. The return value is ignored.
    //
    // NOT USED in external DLLs
    break;

return TRUE;
}

The double #include directives in the code include function prototypes. The
definition of string variable msg_str, the call of the sprintf function and the call
of the MessageBox function are used to display the following window during
loading the DLL:

and the following window during removing the DLL from the memory:
The window appears every time the external DLL library is loaded into the memory (see description of the LoadExtAlg function below). If you do not want to display the window remove the statements in the function DllMain.

If your controller requires initialisation, insert appropriate statements before the return from the DllMain function in the DLL_PROCESS_ATTACH case statement. For instance, if you want to allocate memory on the global heap call the GlobalAlloc and the GlobalLock functions, etc. (compare Example 1 or 3 in section 6 and in section 8),

The statements below the DLL_PROCESS_DETACH value perform a cleanup of a dynamic-link library (DLL) before the library is unloaded. Use this function to terminate the external controller operation safely. For instance, if you have allocated memory on the global heap call the GlobalUnlock and GlobalFree functions, etc.,

The DllMain function returns a BOOL value which is used only when fdwReason is equal to DLL_PROCESS_ATTACH. The TRUE return value is used to signify that the DLL should remain loaded. The FALSE value is used to signify that the DLL should be immediately unloaded. For all other values of fdwReason, the return value is ignored.

- **ExternalController** - is called by the timer procedure at the beginning of each sample period and contains "the body" of the controller. The code of the ExternalController function is executed during interrupt time. See section 6 for implementation details,

- **ExternalIPC** - function used to exchange information between an external DLL library and MATLAB/Simulink environment. This function is called by the interface software procedure hl_call from MATLAB, so it is not executed during the interrupt time. See section 8 for implementation details,

- additional user-defined application-dependent functions.
3.1 Code and data segments limitations

The procedure of an external controller is located in the external library and is executed during interrupt time. The external controller can call only the following API functions: PostMessage, timeGetSystemTime, timeGetTime, timeSetEvent, timeKillEvent, midiOutShortMsg, midiOutLongMsg, and OutputDebugStr.

The external inter-process procedure is not executed during interrupt time, so it may call any system functions.

3.2 Creating 32-bit DLL libraries

In this section we explain how to compile and link DLL libraries using Borland C++ v.5.0, WATCOM C/C++ v.10.6 and Microsoft Visual C++ ver.2.0 compilers. If you want to use another compiler consult the documentation for proper use of the compiler and linker.
3.3 Borland C++ ver. 5.0

To obtain a 32-bit DLL executable file execute the following command (see the mk_bcc.bat file on the installation disk):

```bash
rem Batch file for Borland C v.5.0
rem Usage: mk_bc32 file_without_extension
rem The files with C extension will be compiled

rem Set the root directory to the compiler
set BCR=J:\BC5

%BCR%\bin\Bcc32 -P- -c -I%BCR%\INCLUDE -DDllMain=DllEntryPoint %1.c

%BCR%\bin\Tlink32 -L%BCR%\LIB -Tpd -aa -c -x
%BCR%\LIB\c0d32.obj
    %1.obj, %1, %1,
%BCR%\LIB\import32.lib %BCR%\LIB\cw32.lib
```

Replace the definition of the $\textit{BCR}$ variable by the path to root directory of the Borland C++ compiler. Replace the $\%1$ by the name of your C-code source file (without extension).

The presented commands create a Windows 32-bit executable DLL file.
3.3.1 WATCOM C/C++ v.10.6

The following batch file compiles and links 32-bit DLL library (see the \texttt{mk\_wcc\_bat} file):

```
rem Batch file for Watcom C v.10.6/11.0
rem Usage: mk_wc32 file\_without\_extension
rem The files with C extension will be compiled

rem Set the root directory to the compiler
set WC_ROOT=h:\lang\watcom;

wcc386 %1.c -iWC_ROOT\h;WC_ROOT\h\nt -w4 -e25
    -zq -otexan -bd -fp3 -4r -bt=nt -mf

wlink SYS nt\_dll op m op maxe=25 op q op symf N %1 F %1.obj

wlib -n -b %1.lib +%1.dll
```

The compiler \texttt{wcc386} compiles the \%1.c file. Replace the definition \texttt{WC\_ROOT} environmental variable by the name of WATCOM C/C++ root directory. Replace the \%1 string by the name of your C-file.
3.3.2 Microsoft Visual C++ v. 2.0

To obtain a 32-bit DLL executable file execute the following command (see the mk_mcvc.bat file on the installation disk):

```rem
Batch file for Microsoft Visual C v.2.0
rem Usage: mk_mc32 file_without_extension
rem The files with C extension will be compiled

rem Set the root directory to the compiler
set MC_ROOT=g:\apps\msvc20

%MC_ROOT%\bin\cl /nologo /MT /W3 /GX /YX /O2 /D "__export=" /D "WIN32" /D "NDEBUG" /D "_WINDOWS"
/I%MC_ROOT%\include /c %1.c

@echo %MC_ROOT%\lib\kernel32.lib >> mc_opts.lnk
@echo %MC_ROOT%\lib\user32.lib >> mc_opts.lnk
@echo %MC_ROOT%\lib\libcmt.lib >> mc_opts.lnk
@echo %MC_ROOT%\lib\oldnames.lib >> mc_opts.lnk
@echo /NOLOGO /SUBSYSTEM:windows /DLL >> mc_opts.lnk
@echo /INCREMENTAL:no /MACHINE:I386 >> mc_opts.lnk
@echo /EXPORT:_ExternalController@84 >> mc_opts.lnk
@echo /EXPORT:_ExternalIPC@12 >> mc_opts.lnk

%MC_ROOT%\bin\link @mc_opts.lnk %1.obj
```

Replace the definition of the MC_ROOT variable by the path to root directory of the Visual C++ compiler. Replace the %1 string by the name of your C-code source file (without extension).
4. Exported Functions

Two functions from the external DLL must be implemented and exported. The first one is named **ExternalController**. It contains "the body" of the implemented controller. The interrupt handling procedure from RTK calls **ExternalController** at each sampling period. The **ExternalController** procedure is accessed at interrupt time, so it may not call any other operating system procedures except for: `PostMessage`, `timeGetSystemTime`, `timeGetTime`, `timeSetEvent`, `timeKillEvent`, `midiOutShortMsg`, `midiOutLongMsg`, and `OutputDebugStr`.

The second function is named **ExternalIPC**. It is responsible for inter-process communication between external DLL and MATLAB. You may use any system call inside the body of the **ExternalIPC** function because it is not executed during the interrupt time.

**Note:** The C-file containing template C-code for an external DLL library is stored on installation disk as hl_tmf.c.

4.1 ExternalController function

The **ExternalController** function is executed during each sampling period. You can use input arguments passed to this function to calculate control for the D/A converter. The function is called by RTK when:

1) the external DLL library was loaded successfully (use `hl_call( 'LoadExtAlg', DLL_name )` in MATLAB command window) and,

2) the algorithm number was set to 99 (execute `hl_call( 'SetAlgNo', 99 )` in the MATLAB command window).

**Remember that the ExternalController function is executed at the interrupt time. Do not call system procedures except for:** `PostMessage`, `timeGetSystemTime`, `timeGetTime`, `timeSetEvent`, `timeKillEvent`, `midiOutShortMsg`, `midiOutLongMsg`, and `OutputDebugStr`. 
The shell of the *ExternalController* is listed below (see the *hl_tmf.c* file).

```c
void __export __stdcall ExternalController(double *Param,
    double PosVert,      double PosHoriz,
    double SpeedVert,    double SpeedHoriz,
    double RotorVert,    double RotorHoriz,
    double Time,
    double DesValueVert, double DesValueHoriz,
    double *ControlVert, double *ControlHoriz )
{
    double UVert, UHoriz;
    /* Put the source code of your controller here */
    /* Remember that the control values for */
    /* the DC motors must be returned as: */
    /* *ControlVert = UVert; */
    /* *ControlHoriz = UHoriz; */
    UVert  = 0.0;
    UHoriz = 0.0;

    /* Check limits of the control value */
    if( UHoriz >  +1.0 )  UHoriz =  +1.0;
    if( UHoriz <  -1.0 )  UHoriz = -1.0;
    if( UVert  >  +1.0 )  UVert  =  +1.0;
    if( UVert  <  -1.0 )  UVert  = -1.0;

    *ControlVert  =  UVert;
    *ControlHoriz =  UHoriz;
}
```
The `ExternalController` function has twelve input arguments:

- **Param** - pointer to the array containing twenty parameters. The parameters may be set by the `hl_call`('SetP', new_array) function from the MATLAB command window. If the values of the parameters are required in the MATLAB workspace execute the `hl_call`('GetP') command. Use both functions to tune parameters during real-time experiments, in the *on-line* mode,

- **PosVert** - vertical angular position of the beam. The angle is expressed in radians. The position is calculated relative to the origin position set during resetting the encoders. Notice, that before starting your experiments you must execute `hl_call`('ResetEncoder') function,

- **PosHoriz** - horizontal angular position of the beam. The angle is expressed in radians. The position is calculated relative to the origin position set during resetting the encoders,

- **SpeedVert** - vertical angular velocity of the beam expressed in rad/sec. The velocity is calculated according to the following formula:

  \[
  \text{vertical\_angular\_velocity} = \frac{\text{vertical\_angle}\_{\text{current}} - \text{vertical\_angle}\_{\text{previous}}}{\text{sampling\_period}},
  \]

- **SpeedHoriz** - horizontal angular velocity of the beam expressed in rad/sec. The velocity is calculated according to the following formula:

  \[
  \text{horizontal\_angular\_velocity} = \frac{\text{horizontal\_angle}\_{\text{current}} - \text{horizontal\_angle}\_{\text{previous}}}{\text{sampling\_period}},
  \]

- **RotorVert** - velocity of the vertical (main) propeller. This value is obtained from the output of the tachogenerator and is expressed in A/D converter units,

- **RotorHoriz** - velocity of the horizontal (tail) propeller. This value is obtained from the output of the tachogenerator and is expressed in A/D converter units,

- **Time** - time calculated from the beginning of the experiment or from the executing of the `hl_call`('ResetTime') command. Time is expressed in [seconds],

- **DesValueVert** - vertical desired position of the beam. This value can be set by the `hl_call`('SetPWVert', new_des_position) function and get by the call of the `hl_call`('GetPWVert') function,
DesValueHoriz - horizontal desired position of the beam. This value can be set by the \texttt{hl\_call( 'SetPWHoriz', new\_des\_position )} function and get by the call of the \texttt{hl\_call( 'GetPWHoriz') function},

\textbf{ControlVert} - pointer to control value for the vertical (main) DC drive. The control must be calculated by the \texttt{ExternalController} function and assigned to variable pointed to by \texttt{ControlVert} pointer in the expression:

\[
* \text{ControlVert} = \text{new\_control\_value};
\]

The range of control value is $\pm 1.0$. The control equal to +1.0 yields full output action in one direction, and the control equal to -1.0 yields full output action in the opposite direction. The calculated control value is stored in the data buffer located in RTK and may be accessed by the \texttt{hl\_call( 'GetHistory') function},

\textbf{ControlHoriz} - pointer to control value for the horizontal (tail) DC drive. The control must be calculated by the \texttt{ExternalController} function and assigned to variable pointed to by \texttt{ControlHoriz} pointer in the expression:

\[
* \text{ControlHoriz} = \text{new\_control\_value};
\]

The range of control value is $\pm 1.0$. The control equal to +1.0 yields full output action in one direction, and the control equal to -1.0 yields full output action in the opposite direction. The calculated control value is stored in the data buffer located in RTK and may be accessed by the \texttt{hl\_call( 'GetHistory') function}.

In the example given above the \texttt{ExternalController} function does not correspond to any particular controller. It performs only range checking and assigning of control values. The full implementation of a simple controller is shown in Example 1.

The \texttt{ExternalController} function can be called directly from MATLAB environment. In this case the body of the function is not executed at the interrupt time (nonreal-time mode). Such call can be useful as \textit{off-line} testing before starting real-time experiments. See description of the \texttt{GetExtAlgOneStep} function in section 5.
4.2 ExternalIPC function

The *ExternalIPC* function is used to exchange information between the MATLAB workspace and the external DLL library. Typically, it sets the parameters of the external controller or gets the value of any variable located inside the external DLL.

Figure 4.1 illustrates the principle of operation of the external IPC.

![Figure 4-1: ExternalIPC operation principle](image)

An example of the *ExternalIPC* function is shown below (see the *hl_tmf.c* file):

```c
void __export __stdcall ExternalIPC(
    char   *str,
    double *in_arg,
    double *out_arg )
{
```

int i;

if( strcmp( str, "ACTION_1" ) == 0 )
{
    /* Put here code for "ACTION_1" */
    /* Example: return single value */

    *out_arg = *(out_arg + 1) = 1.0;
    *(out_arg + 2) = 1.0;
    return;
}

if( strcmp( str, "ACTION_2" ) == 0 )
{
    /* Put here code for "ACTION_2" */
    /* Example: return 10 elements */

    *out_arg = 1.0;  *(out_arg + 1) = 10.0;
    for( i = 0; i < 10; i++ )
        *(out_arg + 2 + i) = sin( i );
    return;
}

*out_arg = *(out_arg + 1) = 1.0;
*(out_arg + 2) = -1.0;

The *ExternalIPC* function has three input arguments:

- **str** - string variable. Typically, the *str* variable is used to distinguish different actions performed by the *ExternalIPC* function. Actions are initialised by a call of communication functions described in section 5.

if( strcmp( str, "ACTION_1" ) == 0 )
{
    /* Put here code for "ACTION_1" */
return;

if ( strcmp( str, "ACTION_2" ) == 0 )
{
    /* Put here code for "ACTION_1" */
    return;
}

If your ExternalIPC function performs only one action you are not obliged to use this input argument. The str value is one of the input arguments of the CallExtIPC call (see MATLAB function list in section 5),

\textit{in\_arg} - pointer to the array containing input arguments. The input arguments are passed to the function by the CallExtIPC function. The ExternalIPC does not check the size of the \textit{in\_arg} array. The user is responsible for passing appropriate number of input arguments,

\textit{out\_arg} - pointer to the array containing output arguments. The output arguments are passed from the ExternalIPC function to MATLAB workspace. The maximum size of the \textit{out\_arg} array is 4096 double precision floating point elements. The \textit{hl\_call} function during the CallExtIPC call allocates memory available to the variable returned to MATLAB workspace, so it has to be provided with the information about the size of the returned variable (number of rows and number of columns). This information is contained in the first two elements of the \textit{out\_arg} array. The first element is an integer equal to the number of rows. The second element is equal to the number of columns. For instance, if you want to return a matrix with a single row and 10 columns, use the following statements:

\begin{verbatim}
*out_arg = 1.0;        /* Set number of rows */
*(out_arg + 1) = 10.0; /* Set number of columns */
for( i = 0; i < 10; i++ )
    *(out_arg + 2 + i) = sin( i ); /* Fill the matrix */
\end{verbatim}
Be sure to return at least a single element matrix, even if your \textit{ExternalIPC} function does not perform any action:

\begin{verbatim}
*out_arg = 1.0;
*(out_arg + 1) = 1.0; /* Single element matrix */
*(out_arg + 2) = -1.0; /* Return -1.0 */
\end{verbatim}
CHAPTER 5

TWIN ROTOR MIMO SYSTEM

External Interface  Communication with MATLAB. Reference List

5. Communication with MATLAB. Reference List

The functions associated with external DLL library have the following form:

\[
ret = hl\_call(\ 'FunctionName'\ , [\ arguments ] )
\]

where:

- \( ret \) - is the matrix returned from the function call,
- \( hl\_call \) - is the name of the interface between external DLL library and MATLAB environment,
- \( FunctionName \) - is the name of the operation performed by the \( hl\_call \) interface. All the operations associated with external DLL are given in the Table 5.1, and described in the following sections. The operation name is passed as MATLAB string variable. The \( FunctionName \) is not case sensitive,
- \( arguments \) - are optionally arguments required by some actions.

Table 5.1. External DLL operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoadExtLibrary</td>
<td>Load external DLL library to the memory and execute the statements associated with the DLL_PROCESS_ATTACH flag in the DllMain function</td>
</tr>
<tr>
<td>UnloadExtLibrary</td>
<td>Execute the statements associated with the DLL_PROCESS_DETACH flag in the DllMain function and remove the external DLL library from the memory</td>
</tr>
<tr>
<td>GetExtAlgName</td>
<td>Return the name of a previously loaded DLL library</td>
</tr>
<tr>
<td>CallExtIPC</td>
<td>Execute the ExternalIPC procedure</td>
</tr>
<tr>
<td>ExtAlgOneStep</td>
<td>Execute the ExternalController procedure. The execution is initiated by MATLAB and is performed in off-line mode</td>
</tr>
</tbody>
</table>
CHAPTER 5
TWIN ROTOR MIMO SYSTEM
External Interface Communication with MATLAB. Reference List

5.1 Function LoadExtAlg

Purpose: Load external DLL library to the memory.

Synopsis: \[ ret = hl\_call( 'LoadExtAlg', 'dll\_name' ) \]

Arguments:

- \texttt{dll\_name}\ - is the full path and file name of the file containing external DLL library. Typically, the file has the extension DLL. The \texttt{dll\_name} is a string matrix in MATLAB environment. If only the file name is specified the file is searched in:
  - the current directory,
  - the Windows directory (the directory containing WIN.COM),
  - the Windows system directory (the directory containing such system files as GDI.EXE),
  - the directories listed in the PATH environment variable.

Description: This function is called to load a DLL library containing an external controller and an external inter-process communication procedure. The loaded DLL must export the \texttt{ExternalController} and \texttt{ExternalIPC} functions. After the DLL is loaded the \texttt{DllMain} function is started immediately. The function returns 1 if library is loaded successfully. The window shown below is displayed by RTK. In the case of any fault the error message is displayed in MATLAB command window or in the message box.
The following error messages can be displayed in the MATLAB command window:

LoadExtAlg function requires two arguments - the LoadExtAlg function requires two input arguments: the first is function name and the second is the name of external DLL,

LoadExtAlg function requires that the second argument is a string - the external DLL library name must be a string.

After displaying an error message in the MATLAB command window the execution of the function is terminated immediately.

The following messages may appear in the RTK message window:

External library loaded. Unload it first - the external DLL library is already loaded. You must remove the library from the memory first. Use the UnloadExtAlg function and next try to load library once more,

Can not load xxxxx library - can not load DLL executable xxxxx. Consider one of the following reasons for the error:

• system is out of memory, executable file is corrupt, or reallocations are invalid,
• file xxxxx is not found,
• path is not found,
• there is a sharing or network-protection error,
• application is designed for a different operating system,
• type of executable file is unknown,
• attempt is made to load a real-mode application developed for an earlier version of Windows,

Can not find ExternalController function - there is no ExternalController function exported,

Can not find ExternalIPC function - there is no ExternalIPC function exported.

To execute the control algorithm implemented as the ExternalController function included in loaded DLL library in real-time you must set algorithm number in RTK to 99. For this purpose use the command:

```
ret = hl_call( 'SetAlgNo', 99 );
```
Example:

To load hl_tmf.dll executable located in the D:\MATLAB\DLL directory execute the following command:

```
ret = hl_call( 'LoadExtAlg', 'D:\MATLAB\DLL\HL_TMF.DLL' )
```
5.2 Function UnLoadExtAlg

Purpose: Remove the external DLL library from the memory.

Synopsis: \( \text{ret} = \text{hl}\_\text{call}( '\text{UnLoadExtAlg}' ) \)

Arguments:

\( \text{none} \).

Description: This function removes previously loaded external DLL from the memory. Before removing the \text{DllMain} function is executed. If the number of control algorithm in the RTK was equal to 99 then the number of control algorithm is set to zero. The function always returns 1.

Example:

To remove the external DLL executable from the memory execute the following command:

\( \text{ret} = \text{hl}\_\text{call}( '\text{UnLoadExtAlg}' ) \)
5.3 Function GetExtAlgName

**Purpose:** Returns the full path and the name of loaded external DLL executable.

**Synopsis:**

```
ret = hl_call( 'GetExtAlgName' )
```

**Arguments:**

`none`.

**Description:** This function returns a string matrix containing the full path and the name of a previously loaded DLL library. The return value is equal to the second argument of the `LoadLibrary` function. An empty matrix is returned if there is no external library loaded or if the library was removed from the memory.

**Example:**

After executing the command:

```
ret = hl_call( 'LoadExtAlg', 'D:\MATLAB\DLL\HL_TMF.DLL' );
```

the command:

```
ret = hl_call( 'GetExtAlgName' );
```

sets the string ‘D:\MATLAB\DLL\HL_TMF.DLL’ to variable `ret`. After commands:

```
ret = hl_call( 'UnloadExtAlg' );
ret = hl_call( 'GetExtAlgName' );
```

the `ret` variable is an empty matrix.
5.4 Function CallExtIPC

**Purpose:** Executes the \textit{ExternalIPC} procedure.

**Synopsis:** \texttt{ret = hl\_call( 'CallExtIPC', subfunction, input\_argument )}

**Arguments:**

\textbf{subfunction} - string passed to the \textit{ExternalIPC} function as the first argument, usually used to distinguish different actions performed by this function,

\textbf{input\_arguments} - data passed to the \textit{ExternalIPC} function as the second argument. Notice: the \textit{ExternalIPC} function does not check the size of \textit{input\_arguments} matrix. You must pass appropriate number of elements in the input matrix. \textbf{Matrix containing at least one element must be passed as the third input argument.}

**Description:** This function calls the \textit{ExternalIPC} function located in the external DLL library. Two input arguments are passed to the function - one string and one matrix. Before the \textit{ExternalIPC} function is executed the \texttt{hl\_call} function allocates the array of 4096 double precision elements for output arguments. The output values are returned to MATLAB workspace as the \texttt{ret} variable. This function provides a way to exchange data between MATLAB workspace and external DLL library. Typically, this function is used to set parameters of the external controller or to read the value of any variable located in the external DLL module.
Example:

Let us consider the *ExternalIPC* function containing the following C-code:

```c
if( strcmp( str, "ACTION_1" ) == 0 )
{
    *out_arg = 1.0;                /* Set number of rows */
    *(out_arg + 1) = 10.0;         /* Set number of columns */
    for( i = 0; i < 10; i++ )      /* Fill the matrix */
        *(out_arg + 2 + i) = sin( i );
}
```

When you execute the command:

```c
ret = hl_call( 'CallExtIPC', 'ACTION_1', [ 1 ] );
```

the *ret* variable is the single row vector containing 10 elements. The values of elements are equal to the values of sinus function.
5.5 Function ExtAlgOneStep

**Purpose:** Calls `ExternalController` function.

**Synopsis:**  
\[ ret = hl\_call\( 'ExtAlgOneStep', \text{input\_argument} \) \]

**Arguments:**

- `input\_arguments` - the nine-element vector containing:
  - `input\_argument(1)` - vertical angle of the beam [rad],
  - `input\_argument(2)` - horizontal angle of the beam [rad],
  - `input\_argument(3)` - vertical angular velocity of the beam [rad/sec],
  - `input\_argument(4)` - horizontal angular velocity of the beam [rad/sec],
  - `input\_argument(5)` - velocity of the vertical (main) propeller [A/D units],
  - `input\_argument(6)` - velocity of the horizontal (tail) propeller [A/D units],
  - `input\_argument(7)` - time [sec],
  - `input\_argument(8)` - vertical desired position of the beam [rad],
  - `input\_argument(9)` - horizontal desired position of the beam [rad].

**Description:** This function calls the `ExternalController` function located in the external DLL library. Unlike the `ExternalController` function executed by RTK this function does not call the `ExternalController` at interrupt time. The control value is calculated by the `ExternalController` function according to the value of `input\_arguments` and is returned to the MATLAB workspace as `ret` value. Typically, this function is used for *non-real time* testing of an external control algorithm.
Example:

To obtain and plot a control value for different values of vertical angle and vertical angular velocity execute the following commands:

```matlab
i_ind = 1;
for i = -1 : 0.1 : 1
    j_ind = 1;
    for j = -1 : 0.1 : 1
        surf( i_ind, j_ind ) = ... 
        hl_call( 'ExtAlgOneStep', [ i 0 j 0 0 0 0 0 0 ] );
        j_ind = j_ind + 1;
    end
    i_ind = i_ind + 1;
end
```
Example 1: Non-Linear Stabilising Controller with Tuned Coefficients

Let us implement a controller to stabilise the beam of the TRMS model. The assumed control law is:

\[ U_v = VK_v \cdot \text{sign}(x_v) \cdot VK_{v/2} \cdot x_v \cdot x_v, \]

\[ U_h = HK_h \cdot \text{sign}(x_h) \cdot HK_{h/2} \cdot x_h \cdot x_h, \]

where:

- \( U_v, U_h \) - control value for the vertical and horizontal DC drives,
- \( x_v, x_h \) - error between the desired and current vertical/horizontal position of the beam,
- \( VK_v, VK_{v/2}, HK_h, HK_{h/2} \) - feedback coefficients.

All parameters of the controller will be passed as the first argument of the `ExternalController` function.

The C-code source file is shown below (see `hl_nlfb.c` file on the installation diskette). Notice, that the `ExternalIPC` function is not performing any operation.

```c
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <math.h>

#define MODULENAME "ExternalIPC"

char ModuleName[ 128 ];

BOOL WINAPI DllMain(HINSTANCE hDLLInst, DWORD fdwReason,
                     LPVOID lpvReserved)
{
    char msg_str[ 150 ];

    switch (fdwReason)
```
{  
  case DLL_PROCESS_ATTACH:  
    // The DLL is being loaded for the first time by a given process.  
    // Perform per-process initialization here. If the initialization  
    // is successful, return TRUE; if unsuccessful, return FALSE.
    
    GetModuleFileName( hDLLInst, ModuleName, sizeof(ModuleName) - 30 );
    sprintf( msg_str, "External DLL Library: %s", ModuleName );
    MessageBox( (HWND)NULL, "Entry point to external DLL reached",
                  msg_str, MB_OK );
    break;
  }

  case DLL_PROCESS_DETACH:  
    // The DLL is being unloaded by a given process. Do any  
    // per-process clean up here, such as undoing what was done in  
    // DLL_PROCESS_ATTACH. The return value is ignored.  
    // Unload the hook before returning..
    
    sprintf( msg_str, "External DLL Library: %s", ModuleName );
    MessageBox( (HWND)NULL, "Exit point of external DLL reached",
                  msg_str, MB_OK );
    break;

  case DLL_THREAD_ATTACH:  
    // A thread is being created in a process that has already loaded  
    // this DLL. Perform any per-thread initialization here. The  
    // return value is ignored.
    
    break;

  case DLL_THREAD_DETACH:  
    // A thread is exiting cleanly in a process that has already  
    // loaded this DLL. Perform any per-thread clean up here. The  
    // return value is ignored.
break;
}
return TRUE;
}

void __export __stdcall ExternalController(double *Param,
                                      double PosVert,      double PosHoriz,
                                      double SpeedVert,    double SpeedHoriz,
                                      double RotorVert,    double RotorHoriz,
                                      double Time,
                                      double DesValueVert, double DesValueHoriz,
                                      double *ControlVert, double *ControlHoriz )
{
  static double UVert, UHoriz;
  static double err, sgn;
  static double VKsign, VKu2, VUmax, HKsign, HKu2, HUmax;

  VKsign   = Param[ 0 ];
  VKu2     = Param[ 1 ];
  VUmax    = Param[ 2 ];
  HKsign    = Param[ 3 ];
  HKu2     = Param[ 4 ];
  HUmax = Param[ 5 ];

  err = DesValueVert - PosVert;
  sgn = ( err >= 0.0 ) ? 1.0 : -1.0;
  UVert  = VKsign*sgn + VKu2*(err*err);

  err = DesValueHoriz - PosHoriz;
  sgn = ( err >= 0.0 ) ? 1.0 : -1.0;
  UHoriz  = HKsign*sgn * HKu2*(err*err);

  /* Check limits of the control value */
  if( UVert  >  +VUmax )  UVert  = +VUmax;
  if( UVert  <  -VUmax )  UVert  = -VUmax;
if( UHoriz >  \text{+H} \text{Umax} ) \hspace{1em} \text{UHoriz} = \text{+H} \text{Umax} \text{;}

\text{if( UHoriz <  \text{-H} \text{Umax} ) \hspace{1em} \text{UHoriz} = \text{-H} \text{Umax} \text{;}}

*\text{ControlVert} = \text{UVert;}
*\text{ControlHoriz} = \text{UHoriz;}

void \text{\_export \_stdcall ExternalIPC(}
\text{char} \hspace{1em} *\text{str,} \hspace{1em} /* \text{Function} \hspace{1em} */
\text{double} \hspace{1em} *\text{in\_arg,} \hspace{1em} /* \text{Input arguments} \hspace{1em} */
\text{double} \hspace{1em} *\text{out\_arg} \hspace{1em} /* \text{Output values} \hspace{1em} */
\text{)}
\text{
* ( \text{out\_arg} + 0 \hspace{1em} ) = \text{1;}} \hspace{1em} * ( \text{out\_arg} + 1 \hspace{1em} ) = \text{1;}
* ( \text{out\_arg} + 2 \hspace{1em} ) = \text{-1;}
\text{)\text{}}

\text{Now, you can compile your DLL library. To start the experiment enter the following commands:\text\text{:\}}

\text{ret= hl\_call( \text{\textquoteleft loadextalg\textquoteright, \textquoteleft hl\_nlfb.dll\textquoteleft);}}
\text{ret= hl\_call( \text{\textquoteleft setp\textquoteright, [ 0.3 0.1 1 0.4 0.7 1 ] }); \hspace{1em} \% 6 \text{ parameters}}
\text{ret= hl\_call( \text{\textquoteleft setalgno\textquoteright, 99 \text{);}}
7. Example 2: Off-Line Testing of *Externalcontroller*

The following set of commands loads the external DLL library *hl_ss.dll*, sets parameters and calculates the value of control on the angle vs. angular velocity plane. The C-source code of the DLL library is shown in the Example 1.

```plaintext
a = hl_call('loadlibrary');

a = hl_call('loadextalg', 'hl_nlfb.dll');
name = hl_call('getextalgname');
disp([ 'Name: ' name ]);  

p = hl_call('getp');
p(1 : 6) = [ 0.3 0.1 1 0.4 0.7 1 ];
ret = hl_call('setp', p);

surf = [];
i_ind = 1;
for i = -0.1 : 0.007 : 0.1
    j_ind = 1;
    for j = -0.2 : 0.015 : 0.2
        aux = hl_call('ExtAlgOneStep', [ i 0 0 0 0 0 0 j 0 ]);  
        surf( i_ind, j_ind ) = aux(1);
        j_ind = j_ind + 1;
    end
    i_ind = i_ind + 1
end

a = hl_call('unloadextalg');
```


To plot the calculated surface execute the following commands:

\[
y = -0.1 : 0.007 : 0.1;
\]

\[
x = -0.2 : 0.015 : 0.2;
\]

\[
\text{mesh}(x, y, \text{surf});
\]
8. Example 3: PID Controllers Stabilising Velocity of the Propellers

We will implement the PID controller which stabilises vertical and horizontal propeller velocities. There are two independent PID controllers, one for each propeller. The outputs of the PID controllers produce the control value for the D/A converters which are connected to DC drives.

Parameters of the controllers are exchanged using the \textit{ExternalIPC} function.

The controller is described by the following equations.

The proportional part:

\[ e_v = r_v - d_v, \]
\[ e_h = r_h - d_h, \]

where:

\( e_v, e_h \) - errors of the velocities of the vertical and horizontal propellers,
\( r_v, r_h \) - velocities of the vertical and horizontal propellers,
\( d_v, d_h \) - desired values of the velocities.

The integrators:

\[ I_v(t) = K_{i,v} \int_0^t e_v \, dt, \]

if \( I_v > I_{v,\text{SAT}} \) then \( I_v = I_{v,\text{SAT}}, \)

if \( I_v < -I_{v,\text{SAT}} \) then \( I_v = -I_{v,\text{SAT}}, \)

\[ I_h(t) = K_{i,h} \int_0^t e_h \, dt, \]

if \( I_h > I_{h,\text{SAT}} \) then \( I_h = I_{h,\text{SAT}}, \)

if \( I_h < -I_{h,\text{SAT}} \) then \( I_h = -I_{h,\text{SAT}}, \)
where:

\[ K_{I,V}, K_{I,H} \] - are gains of the I parts,

\[ I_{V,sat}, I_{H,sat} \] - are saturation levels of the integrators.

The output from the \text{PID}_{VERTICAL} and \text{PID}_{HORIZONTAL} parts are calculated as:

\[
U_V = K_{P,V} e_V + I_V(t) + K_{D,V} \frac{de_V}{dt},
\]

\[
U_H = K_{P,H} e_H + I_H(t) + K_{D,H} \frac{de_H}{dt},
\]

\[ K_{P,V}, K_{D,V}, K_{P,H}, K_{D,H} \] - are parameters of the proportional and derivative parts of the controller respectively.

Finally, the control for DC drives is bounded to the control limits:

\[
\text{if} \ (U_V < -U_{V,\text{MAX}}) \ \text{then} \ U_V = -U_{V,\text{MAX}},
\]

\[
\text{if} \ (U_V > U_{V,\text{MAX}}) \ \text{then} \ U_V = U_{V,\text{MAX}},
\]

\[
\text{if} \ (U_H < -U_{H,\text{MAX}}) \ \text{then} \ U_H = -U_{H,\text{MAX}},
\]

\[
\text{if} \ (U_H > U_{H,\text{MAX}}) \ \text{then} \ U_H = U_{H,\text{MAX}},
\]

where:

\[ U_{V,\text{MAX}}, U_{H,\text{MAX}} \] - are maximum values of the control.

The C-code source file is shown below (see the \text{hl_rotor.c} file on the installation diskette).

```c
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <math.h>
```
#define IPC_GetParam    "GetParam"   /* Read parameters - 3 values */
#define IPC_SetParam    "SetParam"   /* Set parameters */
#define IPC_GetDesVal   "GetDesVal"  /* Read ref. pos. - 2 values */
#define IPC_SetDesVal   "SetDesVal"  /* Set ref. Pos. - 2 values */

char ModuleName[ 128 ];
double Pars[ 11 ] = { 0.0,      // vertical:    VKp
                      0.0,      // vertical:    VKi
                      0.0,      // vertical:    VKd
                      0.0,      // horizontal:  VIsat
                      0.0,      // horizontal:  VUmax
                      0.0,      // vertical:    HKp
                      0.0,      // vertical:    HKi
                      0.0,      // vertical:    HKd
                      0.0,      // horizontal:  HIsat
                      0.0,      // horizontal:  HUmax
                      0.01      // sample time: Ts
};
double VDesVal = 0.0, HDesVal = 0.0;

BOOL WINAPI DllMain(HINSTANCE hDLLInst, DWORD fdwReason,
                    LPVOID lpvReserved)
{
    char msg_str[ 150 ];
    switch (fdwReason)
    {
    case DLL_PROCESS_ATTACH:
        // The DLL is being loaded for the first time by a given process.
        // Perform per-process initialization here. If the initialization
        // is successful, return TRUE; if unsuccessful, return FALSE.

        GetModuleFileName( hDLLInst, ModuleName, sizeof(ModuleName) - 30 );
        sprintf( msg_str, "External DLL Library: %s", ModuleName );
        MessageBox( (HWND)NULL, "Entry point to external DLL reached",
                    msg_str, MB_OK );
    }
break;

case DLL_PROCESS_DETACH:
    // The DLL is being unloaded by a given process. Do any
    // per-process clean up here, such as undoing what was done in
    // DLL_PROCESS_ATTACH. The return value is ignored.
    // Unload the hook before returning..
    sprintf( msg_str, "External DLL Library: %s", ModuleName );
    MessageBox( (HWND)NULL, "Exit point of external DLL reached",
               msg_str, MB_OK );

break;

case DLL_THREAD_ATTACH:
    // A thread is being created in a process that has already loaded
    // this DLL. Perform any per-thread initialization here. The
    // return value is ignored.

break;

case DLL_THREAD_DETACH:
    // A thread is exiting cleanly in a process that has already
    // loaded this DLL. Perform any per-thread clean up here. The
    // return value is ignored.

break;

return TRUE;

}
TWIN ROTOR MIMO SYSTEM

External Interface Example 3: PID Controllers Stabilising Velocity of the Propellers

double Time,
    double DesValueVert, double DesValueHoriz,
    double *ControlVert, double *ControlHoriz )
{
    static double UVert, UHoriz;
    static double err, err_V_prev, err_H_prev;
    static double IVert = 0.0, IHoriz = 0.0;
    static double VKp, VKi, VKd, VIsat, VUmax,
                    HKp, HKi, HKd, HIsat, HUmax,
                    Ts;

    VKp    = Pars[  0 ];
    VKi    = Pars[  1 ];
    VKd    = Pars[  2 ];
    VIsat  = Pars[  3 ];
    VUmax  = Pars[  4 ];
    HKp    = Pars[  5 ];
    HKi    = Pars[  6 ];
    HKd    = Pars[  7 ];
    HIsat  = Pars[  8 ];
    HUmax  = Pars[  9 ];
    Ts     = Pars[ 10 ];

    /* ----> VERTICAL <----- */
    err  = VDesVal - RotorVert;

    /* Calculate the IVert parts */
    if ( VKi == 0.0 )
        IVert = 0.0; /* Set the I part to zero if VKi == 0 */
    else
        IVert += VKi * err * Ts;
    /* Saturation of the I part */
    if ( VIsat > 0.0 )
    {
        if ( IVert >  VIsat )   IVert =  VIsat;
        if ( IVert < -VIsat )   IVert = -VIsat;
    }
/* Add vertical PID parts: PID = P + I + D */
UVert = VKp*err + IVert + VKd*(err - err_V_prev)/Ts;

err_V_prev = err;

/* ------> VERTICAL <----- */
err = HDesVal - RotorHoriz;

/* Calculate the IVert parts */
if ( HKi == 0.0 )
    IHoriz = 0.0; /* Set the I part to zero if VKi == 0 */
else
    IHoriz += HKi * err * Ts;
/* Saturation of the I part */
if ( HIsat > 0.0 )
{
    if ( IHoriz > HIsat ) IHoriz = HIsat;
    if ( IHoriz < -HIsat ) IHoriz = -HIsat;
}

/* Add vertical PID parts: PID = P + I + D */
UHoriz = HKp*err + IHoriz + HKd*(err - err_H_prev)/Ts;

err_H_prev = err;

/* Check limits of the control value */
if( UVert > +VUmax ) UVert = +VUmax;
if( UVert < -VUmax ) UVert = -VUmax;
if( UHoriz > +HUmax ) UHoriz = +HUmax;
if( UHoriz < -HUmax ) UHoriz = -HUmax;

*ControlVert = UVert;
*ControlHoriz = UHoriz;
void __export __stdcall ExternalIPC(
    char   *str,       /* Function          */
    double *in_arg,    /* Input arguments   */
    double *out_arg )  /* Output values     */
{

    static char function[ 128 ];
    static i;

    strcpy( function, str );
    strupr( function );       /*  Do not care cases  */

    if( strcmp( function, strupr( IPC_SetParam ) ) == 0 )
    {
        for( i=0; i<11; i++ )
            Pars[ i ] = *( in_arg + i );

        *( out_arg + 0 ) = 1;  *( out_arg + 1 ) = 1;
        *( out_arg + 2 ) = 1;
        return;
    }

    if( strcmp( function, strupr( IPC_GetParam ) ) == 0 )
    {
        *( out_arg + 0 ) = 1;  *( out_arg + 1 ) = 11;
        for( i=0; i<11; i++ )
            *( out_arg + i + 2 ) = Pars[ i ];
        return;
    }

    if( strcmp( function, strupr( IPC_SetDesVal ) ) == 0 )
    {
        VDesVal = *( in_arg + 0 );
        HDesVal = *( in_arg + 1 );
        *( out_arg + 0 ) = 1;  *( out_arg + 1 ) = 1;
        *( out_arg + 2 ) = 1;
        return;
    }
}
Note that:

1. in the *ExternalController* we do not use the *Param* input argument. All parameters are passed by the *ExternalIPC* function (the *GetParam* and *SetParam* subfunctions). You must set the parameters of the controller first and start the controller next. Use the following commands:

   \[
   \text{hl}\_\text{call}( \text{'LoadLibrary'} );
   \]
   \[
   \text{hl}\_\text{call}( \text{'LoadExtAlg'}, \text{DLL}\_\text{name} );
   \]
   \[
   \text{par} = \{ \text{your 11 parameter values} \};
   \]
   \[
   \text{hl}\_\text{call}( \text{'CallExtIPC'}, \text{'SetParam'}, \text{par} );
   \]
   \[
   \text{hl}\_\text{call}( \text{'SetAlgNo'}, 99 );
   \]

2. the reference velocities of the propellers are set by the *SetDesVal* function. E.g. the statement:

   \[
   \text{hl}\_\text{call}( \text{'CallExtIPC'}, \text{'SetDesVal'}, [ 200 \ -150 ] );
   \]

   sets the desired velocity of the vertical propeller to 200 and the desired velocity of the horizontal propeller to -150.
There are two Simulink models available - one for the *hl_nlfb* controller and the other for the *hl_rotor* controller. The names of the appropriate Simulink models are *hl_nl* (stored in the *hl_nl.mdl* file) and *hl_rs* (the *hl_rs.mdl* file). The implementation is similar in both cases. This section describes the implementation of the *hl_rs* Simulink model in details.

The main block of the presented Simulink model is *Real Time Task*. It contains the S-function block calling the *hl_rssf.m* S-function (Figure 9-2). *Real Time Task* is exceeded by the signal generators. The signal generators are sources of reference velocities of the propellers. The output of the *Real Time Task* is a vector which contains information about angles of the beam, velocities of the beam, velocities of the propellers, control values for the DC motors and reference positions of the beam. The values of the output vector can be shown on Simulink scopes and are transferred to MATLAB workspace (*To Workspace* block) as the *hist* variable. The *hist* matrix contains additionally time vector (*Clock* block) and reference velocities of the propellers.

Figure 9-1: Simulink model for the *hl_nlfb.dll* library (*hl_rs.mdl* file).
The \textit{hl\_rssrf} file contains the S-function associated with the \textit{Real Time Task} block. The function has one parameter: down sampling ratio. This parameter decreases the rate of output data flow from the output of the \textit{Real Time Task} (Figure 9-3). For example, if the downsampling coefficient is 10, for each 10 sampling periods only one is transferred to the output of the block.

The S-function block included in the \textit{Real Time Task} block contains two parameters: the S-function file name (\textit{hl\_rssrf}) and the downsampling coefficient value (\textit{downsamp}) (Figure 9-4).
The name of the *Real Time Task* block parameter is set during masking of the *Real-Time Task* block. See the *Initialisation commands* field in Figure 9-5.
The block *STOP Practical* block (Figure 9-1) is used to switch off controls for the DC motors. Double clicking the mouse over this block sets control values to zero.

The history of the experiment is transferred to the MATLAB workspace as the *hist* experiment. The statements given below can be used to display the history of the experiment.

```
plot( hist( :, 13 ), hist( :, [ 5 6] ) ); grid; title( 'Rotor velocities' )
```

![](image.png)
plot( hist(:, 13), hist(:, [5 6]), hist(:, 13), hist(:, [11 12])); grid;

\textit{Velocity and Reference Velocity}

plot( hist(:, 13), hist(:, [7 8])); grid; title('Control')
The "body" of the hl_rssf.m S-function file is shown below. Notice that the S-function uses the flag input argument in a different way from the usual.

The flag equal to 0 is used to set the size of the model (the number of continuous states, the number of discrete states, the number of outputs and the number of inputs), to load the external library containing the controller, to set default parameters of the controller, to set the sample period, to start the experiment (set algorithm number to 99) and to set initial value of the auxiliary variables used in the S-function.

The flag equal to 2 is used to set a new value of reference velocities. The value of reference velocities is obtained from the signal generators Vertical and Horizontal (Figure 9-1) and is passed to the S-function as the input argument $u$.

The flag equal to 3 is used to read the history of the experiment from the Real-Time Kernel and transfer it to the output of the Real Time Task block.

The flag equal to 4 is used to calculate the next discrete time point. This flag is used to synchronise the time of the experiment and the time of Simulink model.

The flag equal to 9 is used to terminate the experiment. In this case the algorithm number is set to zero and the external DLL controller is removed from the memory.
function [sys, x0, str, ts ] = sfunc( t, x, u, flag, downsamp )

% S-function hl_pirsf.m

% Global variables
global hl_par history pos_in_history

switch flag,
  case 0, % Initialization
    % 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
    sizes.NumContStates  = 0;
    sizes.NumDiscStates  = 1;
    sizes.NumOutputs     = 10;
    sizes.NumInputs      = 2;
    sizes.DirFeedthrough = 0;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);

    % Get sample period and sample time divider
    T0 = 0.03;
    dummy = hl_call( 'SetSampleTime', T0 );
    dummy = hl_call( 'LoadExtAlg', 'hl_rotor.dll' );
    dummy = hl_call( 'CallExtIPC', 'SetParam', ...
                    [ 0.02 0.01 0.0 1 1 0.01 0.01 0 1 1 T0 ] );
    dummy = hl_call( 'SetAlgNo', 99 );  % external controller
    dummy = hl_call( 'ResetTime' );

    % Set initial values of the control algorithm
    dummy = hl_call( 'SetAlgNo', 0 );
% Wait for the first sample which may be send to the output
while ( hl_call( 'GetNoOfSamples' ) <= downsamp )
    ;
end;
history=hl_call( 'GetHistory' );

% initialize the initial conditions
str = [];       % str is always an empty matrix
ts  = [-2 0];   % initialize the array of sample times
% variable sample time
% Set initial conditions of the state
pos_in_history = 1;
x0 = max( history(:,1) );

case 1, % Unhandled flags
    sys = [];

case 2, % Calculate discrete state
    % Set desired value
    dummy = hl_call( 'CallExtIPC', 'SetDesVal', [ u(1) u(2) ] );
    sys = x;

case 3, % Calculate outputs
    curr_len = size( history, 1 );
    history = history( downsamp + 1 : curr_len, : );

    while ( hl_call( 'GetNoOfSamples' ) + curr_len <= downsamp )
        ;
    end;
    sys = history( downsamp, 1 );

case 4,% Calculate next discrete time point
    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 );

case 9, % Termination
    dummy = hl_call( 'SetAlgNo', 0 );
    dummy = hl_call( 'UnloadExtAlg' );
There is available a GUI interface which may be used for on-line parameter tuning of the `hl_rotor.dll` library. The interface window is called by the `hl_rssl` command. The GUI window contains the edit fields and sliders which can set minimum, maximum and current value of all parameters (Figure 9-6). See the body of the `hl_rssl.m` file for implementation details of the GUI interface.

![GUI Interface](image)

**Figure 9-6:** GUI interface to the `hl_rotor` DLL executable.

The GUI window shown only appears if the `hl_rotor.dll` library is loaded.

It is recommended to start simulation of the `hl_rs` Simulink model first and next execute the `hl_rssl` command. It causes the Simulink model to set new values of the reference velocities for the propellers and gives the ability to change parameters simultaneously.
TWIN ROTOR MIMO SYSTEM
External Interface Example 4: Simulink Model for the External Controller

Notes
## 10. Quick Reference Table

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mk_bcc.bat</td>
<td>batch file for the Borland C/C++ v.5.0 compiler</td>
</tr>
<tr>
<td>mk_msvc.bat</td>
<td>batch file for the Microsoft Visual C/C++ v.2.0 compiler</td>
</tr>
<tr>
<td>mk_wcc.bat</td>
<td>batch file for the Watcom C/C++ v.10.6 compiler</td>
</tr>
<tr>
<td>hl_nlfb.c</td>
<td>example of the C-source code of the external DLL library</td>
</tr>
<tr>
<td>hl_rotor.c</td>
<td>example of the C-source code of the external DLL library</td>
</tr>
<tr>
<td>hl_tmf.c</td>
<td>template C-source code of the external DLL library</td>
</tr>
<tr>
<td>hl_nlfb.dll</td>
<td>compiled version of the hl_nlfb.c file</td>
</tr>
<tr>
<td>hl_rotor.dll</td>
<td>compiled version of the hl_rotor.c file</td>
</tr>
<tr>
<td>hl_nl.mdl</td>
<td>Simulink model for the hl_nlfb.dll DLL library</td>
</tr>
<tr>
<td>hl_rs.mdl</td>
<td>Simulink model for the hl_rotor.dll DLL library</td>
</tr>
<tr>
<td>hl_nlsf.m</td>
<td>S-function for the hl_nl.mdl Simulink model</td>
</tr>
<tr>
<td>hl_rssf.m</td>
<td>S-function for the hl_rs.mdl Simulink model</td>
</tr>
<tr>
<td>hl_rssl.m</td>
<td>GUI interface for the hl_rotor.dll executable</td>
</tr>
</tbody>
</table>