MS150 Modular Servo Workshop - External Interface

33-008-2M5

Feedback

Computer Assisted Learning

Electricity & Electronics

Control & Instrumentation

Process Control

Mechatronics

Robotics

Telecommunications

Electrical Power & Machines

Test & Measurement

Technology Training for tomorrow's world

MS150 Modular Servo Workshop

External Interface

33-008-3M5

Feedback

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

Manual: 33-008-3M5 Ed01 980930 *Printed in England by FI Ltd, Crowborough*

Feedback Part No. 1160–330083M5

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

CAUTION -
RISK OF
ELECTRIC SHOCK

CAUTION -
ELECTROSTATIC
SENSITIVE DEVICE

Refer to accompanying documents

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.

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

Notes

1 Introduction

Although the Real-Time Kernel (RTK) for the MS150 model 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 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 MS151system,
- MATLAB and Simulink environment,
- C programming language,
- basic 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.

This manuals begins with a short description of Real-Time Kernel, presenting both its general structure and principle of data exchange between modules of the system. Then, the manual continues with a discussion of a structure of DLL external library, which is appropriate for creation of external controllers. Next, instructions how to compile and link DLL libraries are given. In section 4 one can find the shells of two basic functions exported from the external DLL: the *ExternalController* and *ExternalIPC* functions for the MS150model are described in details. In section 5 the reference list of function associated with the external DLL library is given. In section 6 an example of external time suboptimal controller is given. Section 7 presents the *off-line* testing of the external controller. In section 8 the model reference adaptive controller is described. Finally, the Simulink model and the S-function for the controller described in section 9 are presented.

NOTE: :example files of the 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 *es_call.dll* file). It contains measurement procedures, digital filters, data acquisition buffer, built-in control algorithms, software to control output analogue channels, 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, a MATLAB program monitors the RTK library, 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 must contain a controller and interface software written in the special form required by the RTK DLL library. In the opposition to the embedded 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 described external interface is that the user does not need to know all implementation details of the real-time program.

The basic knowledge required to implement this technique is given in this manual.

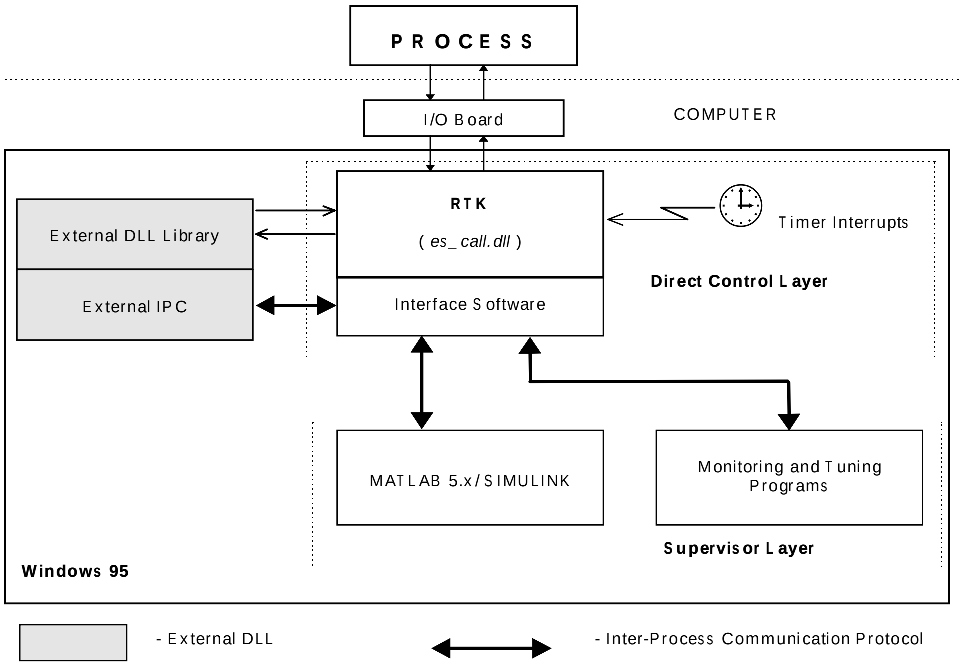


Figure 2-1: General structure of the control system for MATLAB 5 and Windows 95.

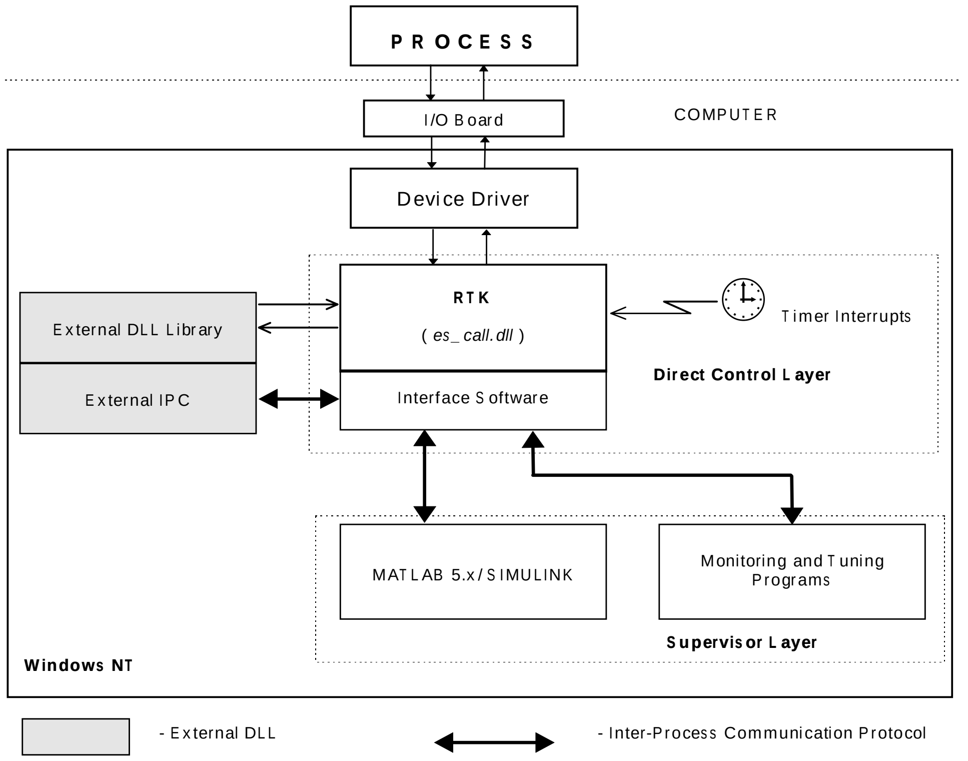


Figure 2-2: General structure of the control system for MATLAB 5 and Windows NT.

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. Next, the external DLL library is loaded.

The external DLL contains two main parts: the controller and the interface procedures. To call the controller from the external DLL the **algorithm number 99** must be selected (see 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 the control value for the DC drive.

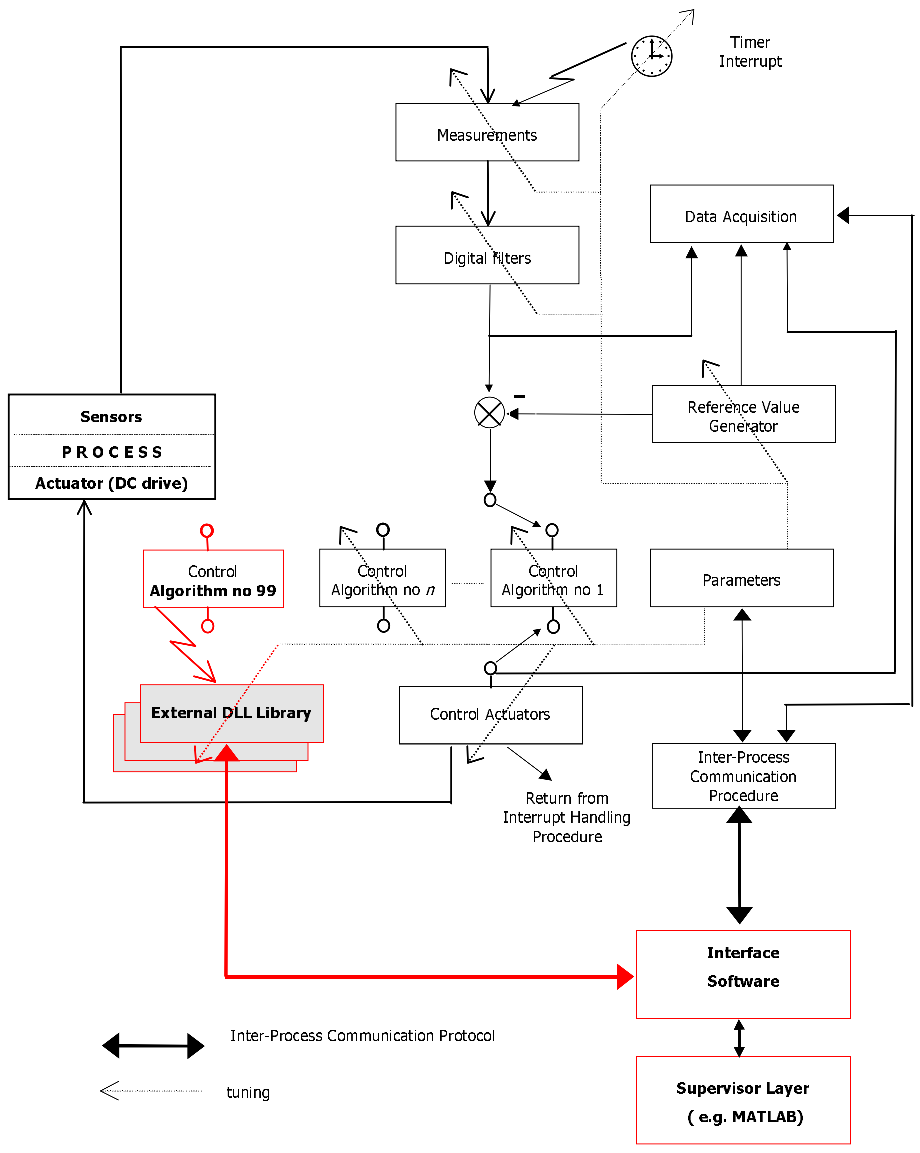


Figure 2-3: Block diagram of RTK.

The external DLL library can be removed from the memory when the experiment is finished. Another external DLL library can be loaded end executed. However, it is not needed to unload the RTK library from the memory when selecting another external DLL. In this sense the external DLL operating mode provides a fast way to test different versions of control algorithms.

3 External Dll Library

There are 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 the development of user-defined controllers.

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

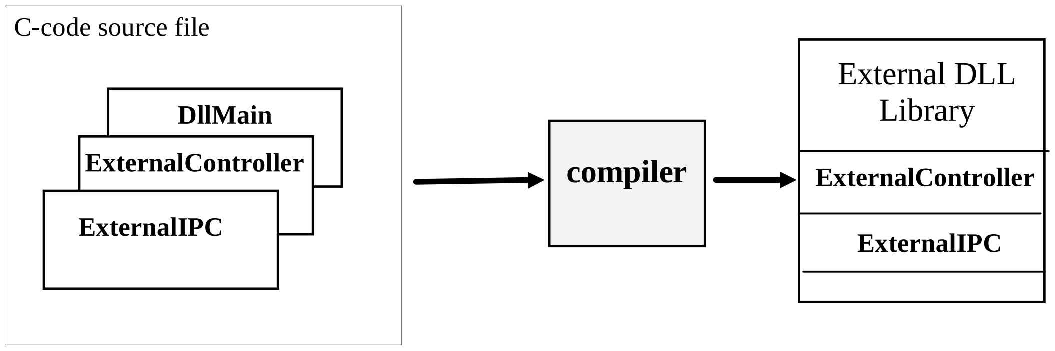


Figure 3-1: Procedure for creating the 32-bit external DLL library

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.

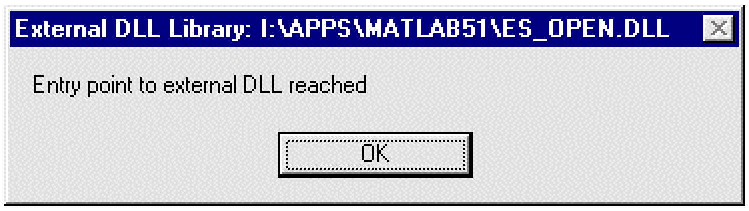
The two last cases are not used by the RTK.

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

```
#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 );
```

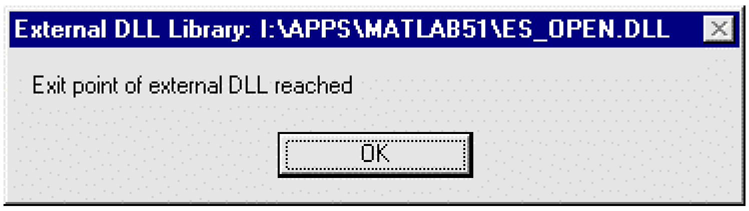
```
        /*                                          */
        /* Insert your termination code here        */
        /*                                          */

               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.
               //
               // 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:


External DLL Library: I:\APPS\MATLAB51\ES_OPEN.DLL

Entry point to external DLL reached

and the following window during removing the DLL from the memory:


External DLL Library: I:\APPS\MATLAB51\ES_OPEN.DLL

Exit point of external DLL reached

The window appears every time the external DLL library is loaded to 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.,

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 is used to exchange information between an external DLL library and MATLAB/Simulink environment. This function is called by the interface software procedure *es_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.

3.2.1. 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):

```
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 *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 Windows 32-bit executable DLL file.

3.2.2. WATCOM C/C++ v.10.6

The following batch file compiles and links 32-bit DLL library (see the *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 *wcc386* compiles the *%1.c* file. Replace the definition *WC_ROOT* environmental variable by the name of WATCOM C/C++ root directory. Replace the *%1* string by the name of your C-file.

3.2.3. 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 *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 *es_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 *es_call('LoadExtAlg', DLL_name)* in MATLAB command window) and,

2) the algorithm number was set to 99 (execute *es_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 *es_tmf.c* file).

```
void __export __stdcall ExternalController(double far *Param,
        double Chan1, double Chan2, double Chan3,
        double Chan4, double Chan5, double Chan6,
        double Time,
        double far *Control)
{
  double U;
  /*                                    */
  /* Put the source code of your controller here */
  /* Remember that the control value for        */
  /* the DC motor must be returned as:          */
  /* *Control = U;                              */

   U = 1.1 * sin( Time ); /* Simple ,open-loop control */

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

  *Control  =  U;

}
```

The *ExternalController* function has nine input arguments:

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

Chan1, *Chan2*, *Chan3, Chan4*, *Chan5* and *Chan6* - current measurements stored in the appropriate elements of the built-in data acquisition buffer. The contents of these variables depends on the last call to the *SetDataSource* function (see *Reference Guide* for details),
Time - time calculated from the beginning of the experiment or from the executing of the *es_call('ResetTime')* command. Time is expressed in [seconds],

Control - pointer to control value for the DC drive. The control must be calculated by the *ExternalController* function and assigned to variable pointed by *Control* pointer in the expression:

**Control = new_control_value;*

The range of control value is ±1.0. The control equal to +1.0 yields a full control signal to the output of the D/A converter in one direction, and the control equal to -1.0 yields full output in the opposite direction. The calculated control value is stored in the data buffer located in RTK and may be accessed by the *es_call('GetHistory')* function.

In the example given above the *ExternalController* function corresponds to open-loop control. It performs only calculation of a sinusoidal control value, range checking and assigning of control value. The full implementation of a controller is shown in section 6.

The *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 *off-line* testing before starting real-time experiments. See description of the *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.

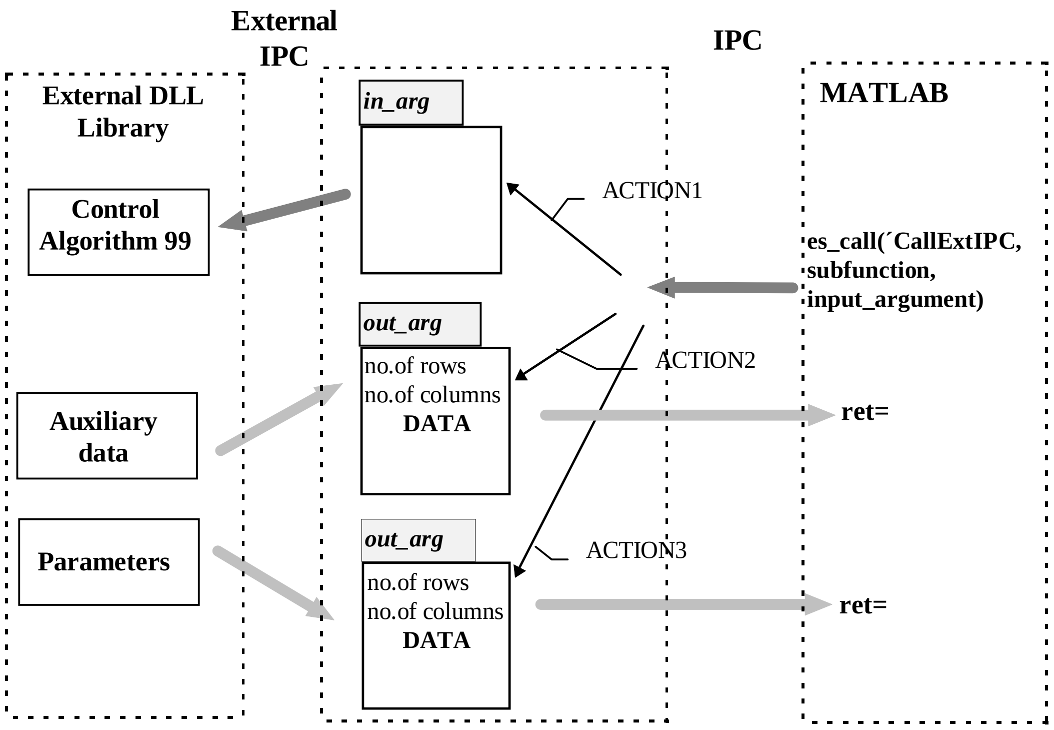


Figure 4-1: External IPC operation principle

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

```
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), *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 *in_arg* array. The user is responsible for passing appropriate number of input arguments,

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 *out_arg* array is 4096 double precision floating point elements. The *es_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 *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:

```
*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 */
```

Be sure to return at least a single element matrix, even if your *ExternalIPC* function does not perform any action:

```
*out_arg = 1.0;
*(out_arg + 1) = 1.0;     /* Single element matrix */
*(out_arg + 2) = -1.0;    /* Return -1.0           */
```

5 Communication With Matlab. Reference List

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

ret = es_call('FunctionName', [arguments])

where:

ret - is the matrix returned from the function call,
es_call - is the name of the interface between external DLL library and MATLAB environment,

FunctionName - is the name of the operation performed by the *es_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 a MATLAB string variable. The *FunctionName* is not case sensitive,

arguments - are optionally arguments required by some actions.

Table 5.1. External DLL operations.

Operation	Description
LoadExtLibrary	Load external DLL library to the memory and execute the statements associated with the *DLL_PROCESS_ATTACH* flag in the *DllMain* function
UnloadExtLibrary	Execute the statements associated with the *DLL_PROCESS_DETACH* flag in the *DllMain* function and remove the external DLL library from the memory
GetExtAlgName	Return the name of the previously loaded DLL library
CallExtIPC	Execute the *ExternalIPC* procedure
ExtAlgOneStep	Execute the *ExternalController* procedure. The execution is initiated by MATLAB and is performed in *off-line* mode

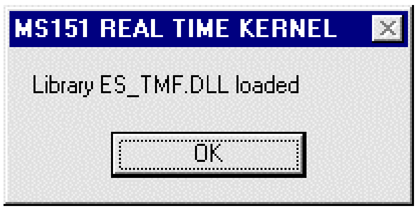
5.1. FUNCTION LOADEXTALG

Purpose: Load external DLL library to the memory.

Synopsis: ***ret = es_call('LoadExtAlg', 'dll_name')***

Arguments: ***dll_name*** - is the file name of the external DLL library. Typically, the file has the extension DLL. The *dll_name* is a string matrix in MATLAB environment. The name of the external DLL library can contain full path to this file or can be stored in on of the directories which are in MATLAB path (see the result of the *matlabpath* function). In both cases the RTK is able to load the external DLL library.

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 *ExternalController* and *ExternalIPC* functions. After the DLL is loaded the *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.


MS151 REAL TIME KERNEL

Library ES_TMF.DLL loaded

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 = es_call( 'SetAlgNo', 99 );
```

Example:

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

```
ret = es_call( 'LoadExtAlg', 'D:\MATLAB\DLL\ES_TMF.DLL' )
```

5.2. FUNCTION UNLOADEXTALG

Purpose: Remove the external DLL library from the memory.

Synopsis: ***ret = es_call('UnloadExtAlg')***

Arguments: ***none***.

Description: This function removes a previously loaded external DLL from the memory. Before removing the *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:
ret = es_call('UnloadExtAlg')

5.3. FUNCTION GETEXTALGNAME

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

Synopsis: ***ret = es_call('GetExtAlgName')***

Arguments: ***none***.

Description: This function returns a string matrix containing the full path and the name of a previously loaded DLL library. 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 = es_call('LoadExtAlg', 'D:\MATLAB\DLL\ES_TMF.DLL');

the command:

ret = es_call('GetExtAlgName');

sets the string '*D:\MATLAB\DLL\ES_TMF.DLL*' to variable *ret.* After commands:

ret = es_call('UnloadExtAlg');
ret = es_call('GetExtAlgName');

the *ret* variable is an empty matrix.

5.4. FUNCTION CALLEXTIPC

Purpose: Executes the *ExternalIPC* procedure.

Synopsis: ***ret = es_call('CallExtIPC', subfunction, input_argument)***

Arguments: ***subfunction*** - string passed to the *ExternalIPC* function as the first argument, usually used to distinguish different actions performed by this function,

input_arguments - data passed to the *ExternalIPC* function as the second argument. Notice: the *ExternalIPC* function does not check the size of *input_argument* matrix. You must pass appropriate number of elements in the input matrix. **Matrix containing at least one element must be passed as the third input argument.**

Description: This function calls the *ExternalIPC* function located in the external DLL library. Two input arguments are passed to the function - one string and one matrix. Before the *ExternalIPC* function is executed the *es_call* function allocates the array of 4096 double precision elements for output arguments. The output values are returned to MATLAB workspace as the *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:

```
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++ )
       *(out_arg + 2 + i) = sin( i ); /* Fill the matrix */
}
```

When you execute the command:

```
ret = es_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 = es_call('ExtAlgOneStep', input_argument)***

Arguments: ***input_arguments*** - the seven-element vector containing:

input_argument(1) - value for the *Chan1* input argument to the *ExternalController* function,
input_argument(2) - value for the *Chan2* input argument to the *ExternalController* function,
input_argument(3) - value for the *Chan3* input argument to the *ExternalController* function,
input_argument(4) - value for the *Chan4* input argument to the *ExternalController* function,
input_argument(5) - value for the *Chan5* input argument to the *ExternalController* function,
input_argument(6) - value for the *Chan6* input argument to the *ExternalController* function,
input_argument(7) - time [sec].

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 *Chan1* and *Chan2* execute the following commands:

```
i_ind = 1;
for i = -1 : 0.1 : 1
  j_ind = 1;
  for j = -1 : 0.1 : 1
   surf( i_ind, j_ind ) = ...
      es_call( 'ExtAlgOneStep', [ i j 0 0 0 0 0 ] );
   j_ind = j_ind + 1;
  end
  i_ind = i_ind + 1;
end
```

6 Example 1: Time Sub-Optimal Controller

The MS151system is described by the following set of linear differential equations:

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

where:

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

K_S, T_S are constants, typically equal to K_S=230 and T_S=0.11,
$x_1$is angle of the DC motor shaft,
$x_2$is angular velocity,
u is input voltage limited to the values from –1.0 to +1.0,
y_1, y_2 are the outputs of the angle and velocity of the DC motor shaft.

The differential equations are:

$$\frac{dx_1}{dt} = x_2,$$
$$\frac{dx_2}{dt} = -\frac{x_2}{T_S} + \frac{K_S}{T_S}u.$$

After dividing the first equation by the second equation we obtain:

$$\frac{dx_1}{dx_2} = \frac{x_2}{-\frac{x_2}{T_S} + \frac{K_S}{T_S}u}$$

and after rearrangement:

$$dx_1 = \frac{x_2 dx_2}{-\frac{x_2}{T_S} + \frac{K_S}{T_S}u}.$$

After integration:

$$\int dx_1 = \int \frac{x_2}{-\frac{x_2}{T_S} + \frac{K_S}{T_S} u} dx_2 .$$

we obtain the formula for the trajectories of the system on the phase-plane:

$$x_1 = -T_S x_2 - T_S K_S u \log(-x_2 + K_S u) + C .$$

The trajectories of the system for maximum and minimum control values are shown in Figure 6-1.

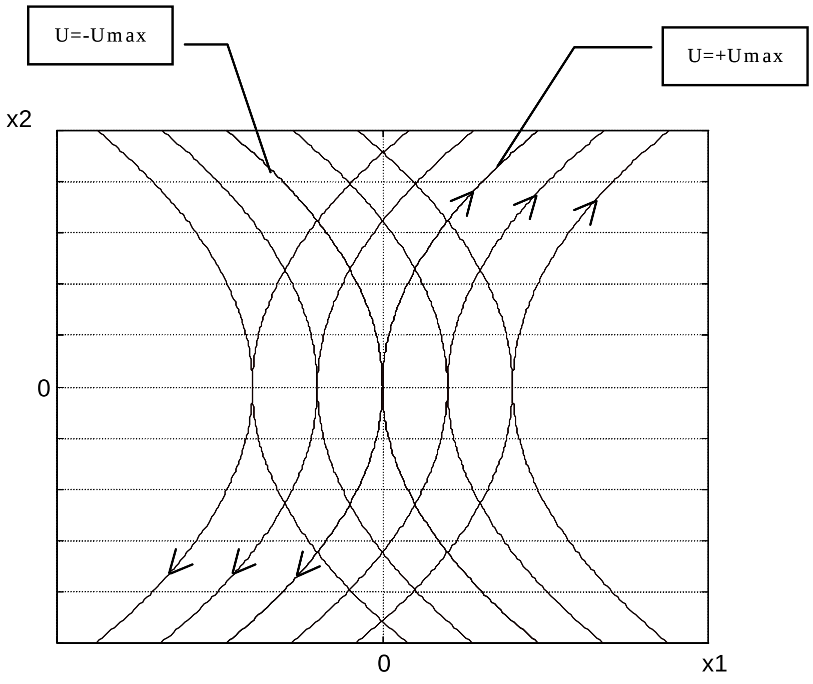


Figure 6-1: Phase-plane trajectories of the DC motor.

Typically, for the time-optimal control policy the branches of the trajectory are used to model the switching curve. The switching curve divides the phase plane into two parts: one for +Umax and other for -Umax control. If the current state is located "above" the switching curve the control value is set to -Umax. If the current state is located "below" the switching curve the control value is set to +Umax.

Below, we will present the time sub-optimal controller for the MS 151 system implemented in the form of the external DLL library. The time sub-optimal controller uses a line instead of the switching curve consisted of the parts of the trajectories of the system.

Such a solution simplifies the implementation and gives the opportunity to observe some new interesting behaviour of the system, like a sliding mode control.

The approximation of the switching curve (switching line) on the phase plane is shown in Fig.6.2.

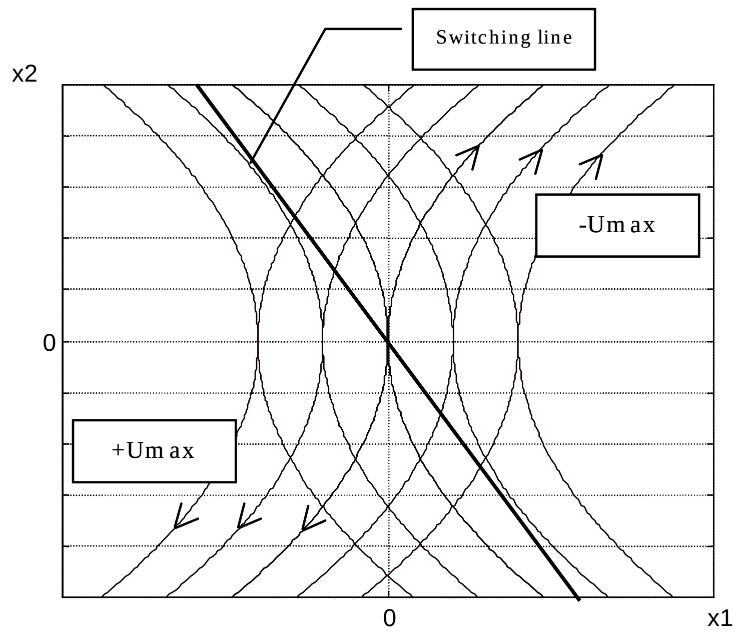


Figure 6-2: Approximation of the switching curve.

The switching line is calculated due to the following formula:

$$x_2 = Ax_1$$

where *A* is a constant.

The C-code source file is shown below (see *es_sto.c* file). Notice, that the *ExternalIPC* function is used to set and read two parameters: the parameter of the switching line *A* and maximum value of the control *Umax*.

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


double A;     // switching line parameter
double UMax;  // maximum control

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 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 far *Param,
         double Chan1, double Chan2, double Chan3,
         double Chan4, double Chan5, double Chan6,
         double Time, double far *Control)
{
   //
   // Chan1 - position
   // Chan2 - velocity
   // Chan3 - from internal built-in signal generator
   //

   double err;

   err = Chan1 - Chan3;
   if ( Chan2 > A*err )
     *Control = -UMax;
   else
     *Control = +UMax;
}


void __export __stdcall ExternalIPC(
                    char   *str,       /* Function          */
                    double *in_arg,    /* Input arguments   */
                    double *out_arg )  /* Output values     */
{
  char function[ 128 ];

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

```
    if( strcmp( function, strupr( "SetParam" ) ) == 0 )
    {
      A    = *( in_arg + 0 );
      UMax = *( in_arg + 1 );
      *( out_arg + 0 ) = 1;  *( out_arg + 1 ) = 1;
      *( out_arg + 2 ) = 0;
      return;
    }
    if( strcmp( function, strupr( "GetParam" ) ) == 0 )
    {
      *( out_arg + 0 ) = 1;  *( out_arg + 1 ) = 2;
      *( out_arg + 2 + 0 ) = A;
      *( out_arg + 2 + 1 ) = UMax;
      return;
    }

    *( out_arg + 0 ) = 1;  *( out_arg + 1 ) = 1;
    *( out_arg + 2 ) = -1;  /* Always returns -1.0 */
  }
```

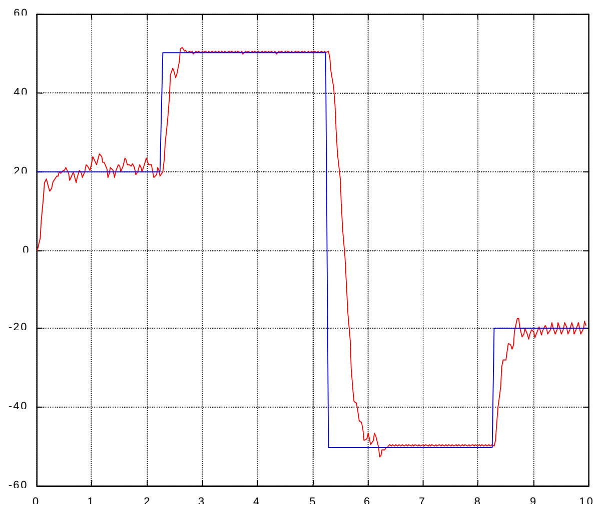
Now, you can compile your DLL library. To start the experiment enter the following commands:

```
ret = es_call('LoadExtAlg', 'es_sto.dll');
ret = es_call('CallExtIPC', 'SetParam', [ -10 1 ] ); % 2 parameters
ret = es_call('SetDataSource', [ 8 9 13 4 5 6 ] );
ret = es_call('ResetEncoder' );
ret = es_call('SetPW', [1 3 3 3 3 -50 -20 20 50 zeros( 1, 11) ] );
ret = es_call('SetSampleTime', 0.03);
ret = es_call('SetAlgNo', 99 );
pause( 10 );
h = es_call('GetHistory' );
ret = es_call('UnloadExtAlg' );
```

The results are shown below.

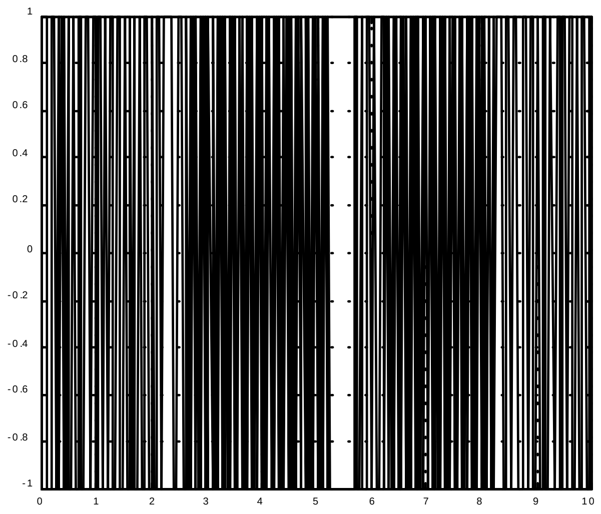
To plot the angle and the reference value execute the following command:

```
plot(h(1,:)-h(1,1),h([2 4],:));grid
```


60
40
20
0
-20
-40
-60
0
1
2
3
4
5
6
7
8
9
10

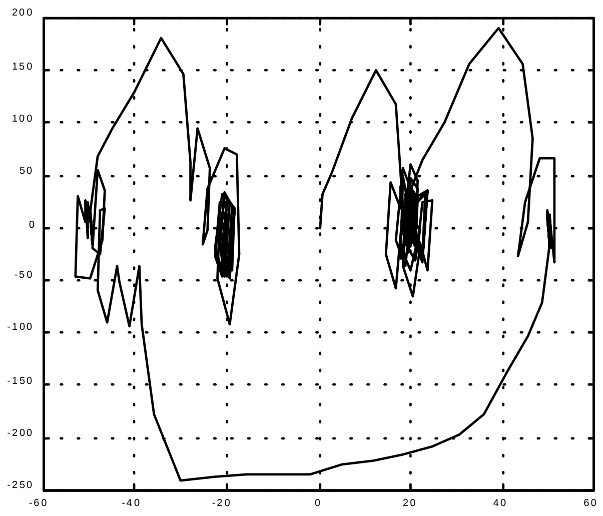
To plot control value execute:

plot(h(1,:)-h(1,1),h(8,:));grid



The following command plots the phase-plane diagram:

plot(h(2,:),h(3,:));grid



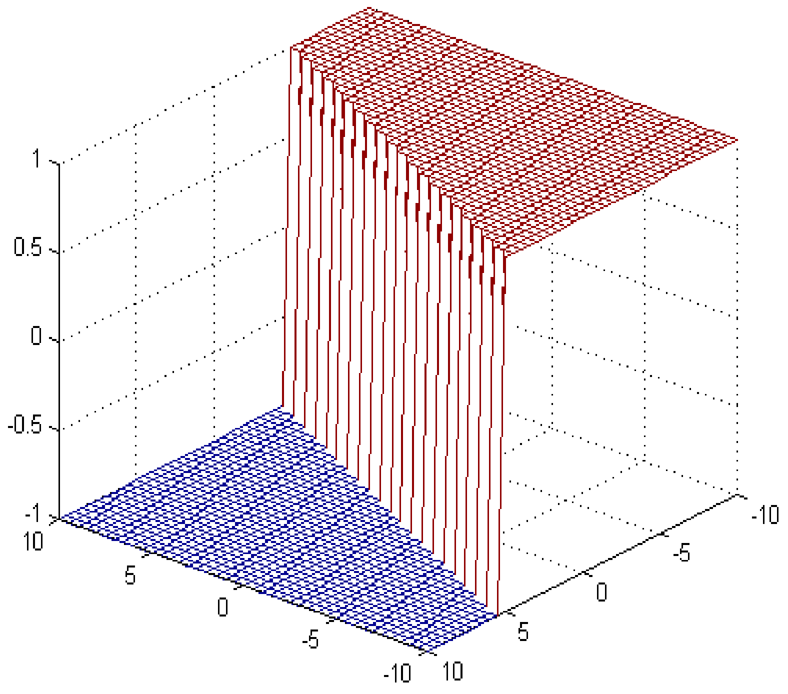
7 Example 2: Off-Line Testing Of *Externalcontroller*

The following set of commands loads the external DLL library *es_sto.dll* , sets parameters and calculates the value of control on the angle (x_1) vs. angular velocity plane(x_2). The C-source code of the DLL library was shown in Example 1.

```
a= es_call('loadextalg', 'es_sto.dll');
name = es_call('getextalgname');
disp( [ 'Name: ' name ] );

ret = es_call( 'CallExtIPC', 'SetParam', [-2 1] );

surf = [];
i_ind = 1;
for i = -10 : 0.5 : 10
  j_ind = 1;
  for j = -10 : 0.5 : 10
   aux = es_call( 'ExtAlgOneStep',[ i j 0 0 0 0 0 ]);
   surf( i_ind, j_ind ) = aux( 1 );
    j_ind = j_ind + 1;
  end
  i_ind = i_ind + 1
end

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

To plot the calculated surface execute the following commands:

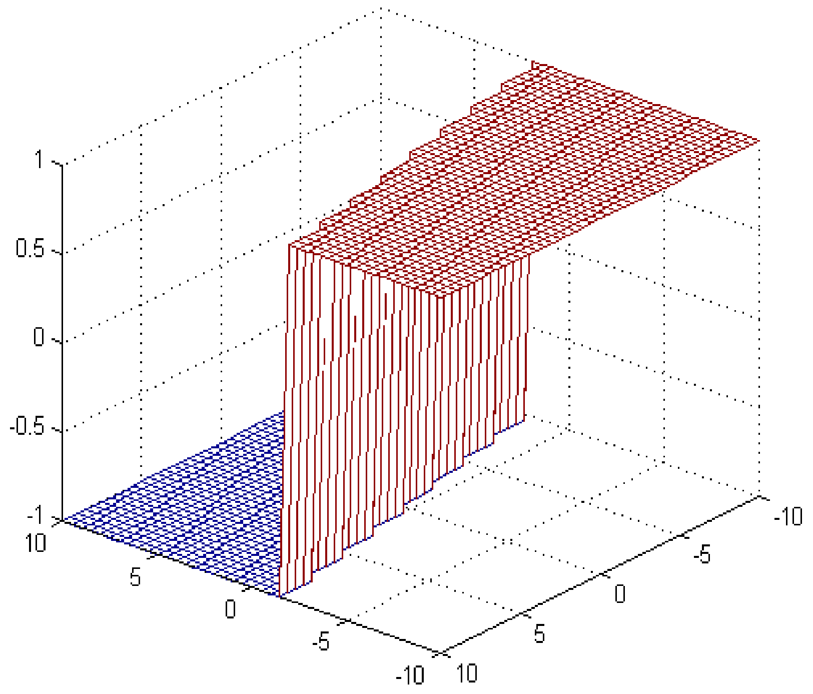
```
y = -10 : 0.5 : 10;
x = -10 : 0.5 : 10;
mesh( x, y, surf );
view( -140, 30 )
```



After the following commands:

```
a= es_call('loadextalg', 'es_sto.dll');
name = es_call('getextalgname');
ret = es_call( 'CallExtIPC', 'SetParam', [-0.2 1] );

surf = [];
i_ind = 1;
for i = -10 : 0.5 : 10
  j_ind = 1;
  for j = -10 : 0.5 : 10
   aux = es_call( 'ExtAlgOneStep',[ i j 0 0 0 0 0 ]);
   surf( i_ind, j_ind ) = aux( 1 );
    j_ind = j_ind + 1;
  end
  i_ind = i_ind + 1
end

a= es_call('unloadextalg');

y = -10 : 0.5 : 10;
x = -10 : 0.5 : 10;
mesh( x, y, surf );
view( -140, 30 )
```

we obtain the following plot:



Notes

8 Example 3: Model Reference Adaptive Control (MRAC) r

One mode of operation for a servomotor is to trace the desired value. Standard controllers use fixed settings and if the conditions of its operation have been changed the new parameters must be set - usually by operator. The advanced idea is to tune parameters of the controller automatically.

8.1. A SHORT VIEW OF MRAC THEORY

Model reference adaptive control, an explicit adaptive technique, has been very attractive from the very beginning of the adaptive control approach.

The basic idea of the method is illustrated in Figure 8.1, where a reference model is used to specify the desired performances of the basic loop, consisting of the controlled process and a controller.

The reference model consists of the servo model and the same type of the controller as used in the basic loop. In Figure 8-1 the classical feedback closed-loop controller is used in the basic loop, but in some cases feedback controllers and pre-filters can be used as well.

The current output of the controlled process is compared with the reference model output and the parameters of the controller are modified in such a way that the response of the controlled process follows that of the reference model.

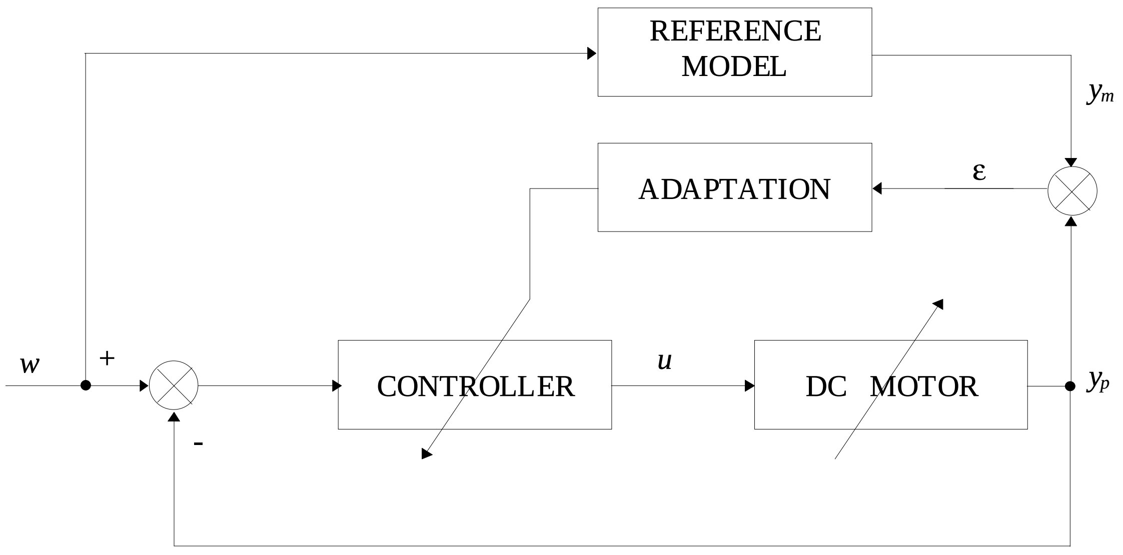


Figure 8-1: Block diagram of a model reference adaptive system (MRAC)

The criterion to be minimized is:

$$I = \frac{1}{2}\int_{t}^{t+\Delta t} \varepsilon^2(\tau)d\tau$$

where:

$$\varepsilon(\tau) = y_p(t) - y_m(t)$$

is the difference of the controlled process outputs and reference model outputs. The adjustment of the classical controller parameters can be made using the steepest descent technique.

8.2. ADAPTIVE CONTROLLER DESIGN

The rewritten transfer function of the DC motor is given by:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{\frac{K_s}{T_s}}{s^2 + \frac{1}{T_s}s}$$

To control the DC motor we use the controller operating according to law:

$$u = fw - q_0\dot{y} - q_1 y$$

There are three parameters:

f proportional to desired value
q_0 proportional to servo-motor velocity
q_1 proportional to servo-motor position

Figure 8-2 shows the basic loop with implemented controller.

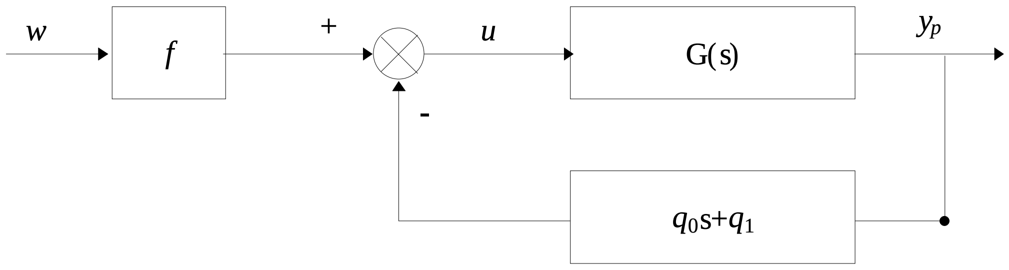


Figure 8-2: The applied control loops

where :

w - desired value
u - control,
y - output (position and velocity),
f,q_0,q_1 - parameters of the controller.

This kind of the controller is easy to implement while designing the adaptive controller. In this case the equation of the controller in the reference model is given by:

$$u_m = f_m w - q_{0m}\dot{y}_m - q_{1m} y_m$$

and the equation of the controller in the basic loop (the real DC motor is controlled) is given by:

$$u = f_p w - q_{0p}\dot{y} - q_{1p} y$$

The transfer function of the reference model is given by:

$$G_m(s) = \frac{y_m}{w} = \frac{f_m \frac{K_m}{T_m}}{s^2 + \left(\frac{1}{T_m} + \frac{K_m}{T_m} q_{0m}\right)s + \frac{K_m}{T_m} q_{1m}}$$

The transfer function of the basic loop is given by:

$$G_w(s) = \frac{y_p}{w} = \frac{f_p \frac{K_p}{T_p}}{s^2 + \left(\frac{1}{T_p} + \frac{K_p}{T_p} q_{0p}\right)s + \frac{K_p}{T_p} q_{1p}}$$

Assume that we attempt to change the parameters of the controller so that the error ε between outputs of the process and the reference model is driven to zero. To make the value of the criterion *I* small it is reasonable to change the parameters in the direction of the negative gradient of *I*.
Applying this concept for the first parameter leads to:

$$\Delta f_p = -g_{f_p} grad_{f_p} I = -g_{f_p} \frac{\partial I}{\partial f_p}$$

For changing speed of the first parameter variations it follows that:

$$\frac{df_p}{dt} = -g_{f_p} \frac{\partial}{\partial t} \frac{\partial I}{\partial f_{f_p}} = -g_{f_p} \frac{\partial}{\partial f_p} \frac{\partial I}{\partial t}$$

If the performance criterion is introduced:

$$\frac{df_p}{dt} = -g_{f_p} \frac{\partial I}{\partial f_{f_p}} \varepsilon^2(t) = -2g_{f_p} \varepsilon(t) \frac{\partial \varepsilon}{\partial f_p} = -2g_{f_p} \varepsilon(t) \frac{\partial y_p}{\partial f_p}$$

Integrating leads to:

$$f_p = -g_{f_p} \int_0^t \varepsilon(t) \frac{\partial y_p}{\partial f_p} dt + f_p(0)$$

For remaining parameters we obtain:

$$q_{0p} = -g_{q_{0p}} \int_0^t \varepsilon(t) \frac{\partial y_p}{\partial q_{0p}} dt + q_{0p}(0)$$

$$q_{1p} = -g_{q_{1p}} \int_0^t \varepsilon(t) \frac{\partial y_p}{\partial q_{1p}} dt + q_{1p}(0)$$

where the sensitivity functions can be obtained by corresponding derivations and suitable approximations as follows:

$$\frac{\partial y_p}{\partial f_p} = \frac{\partial}{\partial f_p}\left(G_w(s)w\right) = \frac{\frac{K_p}{T_p}}{s^2 + \left(\frac{1}{T_p} + \frac{K_p}{T_p} q_{0p}\right)s + \frac{K_p}{T_p} q_{1p}} w = \frac{y_p}{f_p} \approx \frac{K_p T_m}{f_m K_m T_p} y_m$$

$$\frac{\partial y_p}{\partial q_{0p}} = \frac{\partial}{\partial q_{0p}}\left(G_w(s)w\right) = -\frac{f_p\left(\frac{K_p}{T_p}\right)^2 s}{\left(s^2 + \left(\frac{1}{T_p} + \frac{K_p}{T_p} q_{0p}\right)s + \frac{K_p}{T_p} q_{1p}\right)^2} w =$$

$$= -\frac{s\frac{K_p}{T_p}}{s^2 + \left(\frac{1}{T_p} + \frac{K_p}{T_p} q_{0p}\right)s + \frac{K_p}{T_p} q_{1p}} y_p \approx -\frac{K_p T_m}{f_m K_m T_p} G_m(s) \cdot s \cdot y_m$$

$$\frac{\partial y_p}{\partial q_{1p}} = \frac{\partial}{\partial q_{1p}}\left(G_w(s)w\right) = -\frac{f_p\left(\frac{K_p}{T_p}\right)^2}{\left(s^2+\left(\frac{1}{T_p}+\frac{K_p}{T_p}q_{0p}\right)s+\frac{K_p}{T_p}q_{1p}\right)^2}w =$$

$$= -\frac{\frac{K_p}{T_p}}{s^2+\left(\frac{1}{T_p}+\frac{K_p}{T_p}q_{0p}\right)s+\frac{K_p}{T_p}q_{1p}}y_p \approx -\frac{K_pT_m}{f_mK_mT_p}G_m(s)y_m$$

We do not know the actual parameters of the DC motor, hence we can assume the constants determining the rate of gradient adaptation $g_{q_{frp}}, g_{q_{0rp}}, g_{q_{1rp}}$ as:

$$g_i = \overline{g}_i\frac{f_mK_mT_p}{K_pT_m}$$

In this case the parameters of the controller are given by:

$$f_p = -\overline{g}_{f_p}\int_0^t \varepsilon(t)y_m dt + f_p(0)$$

$$q_{0p} = \overline{g}_{q_{0p}}\int_0^t \varepsilon(t)\cdot s\cdot G_m(s)\cdot y_p dt + q_{0p}(0)$$

$$q_{1p} = \overline{g}_{q_{1p}}\int_0^t \varepsilon(t)\cdot G_m(s)\cdot y_p dt + q_{1p}(0)$$

8.3. MODEL REFERENCE ADAPTIVE CONTROLLER

The MRAC algorithm was implemented using the EXTERNAL INTERFACE.

To run the adaptive controller in MATLAB Command Window type: es_mracm (see Figure 8-2).

The *es_mracsf.m* file contains the S-function associated with the *Real Time Task* block.

The function has six parameters:

- *Reference source* - this parameter is set by selecting the appropriate element in the listbox which contains three fields:
  - ◊ *Simulink generator* - this option causes the Simulink signal generator *Desired Position* to be the source of the reference angle (see Figure 8-4),
  - ◊ *External signal* - the reference signal comes from the MS151model,
  - ◊ *Built-in generator (square)* - the signal generator built into the RTK is the source of the square wave which determines the reference value,
- *Reference model parameters [Ks Ts]* - Ks and Ts parameters of the reference model,
- *Reference model controller parameters [f q0 q1]* - Parameters to control the reference model,
- *Constants rate of gradient adaptation [gf gq0 gq1]* - constants rate of gradient adaptation
- *Max control* - maximum value of the control,
- *Downsampling ratio* - this parameter decreases the rate of output data flow from the output of the *Real Time Task* (see Figure 8-5). For example, if the downsampling coefficient is 10, for each 10 sampling periods only one is transferred to the output of the block.

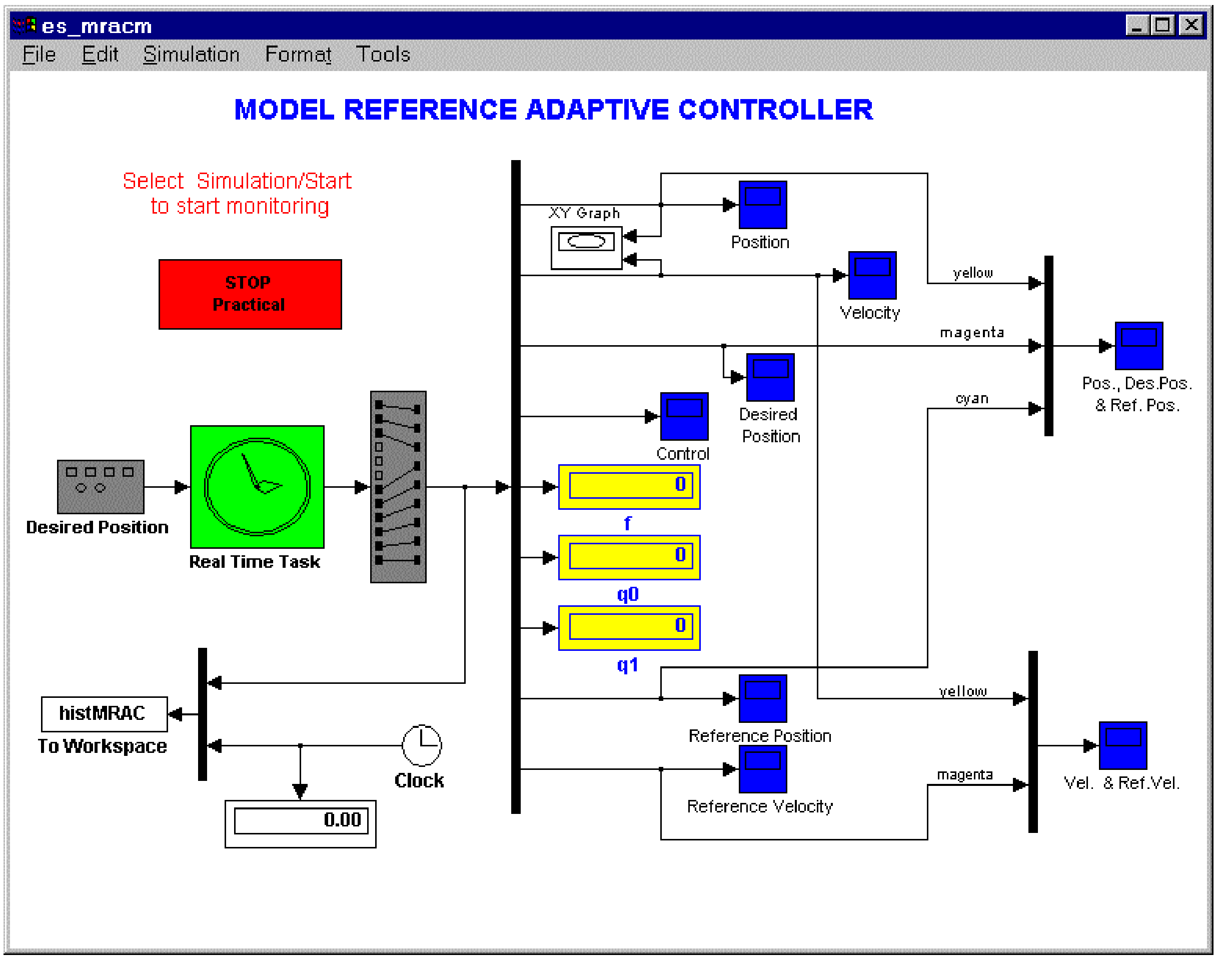
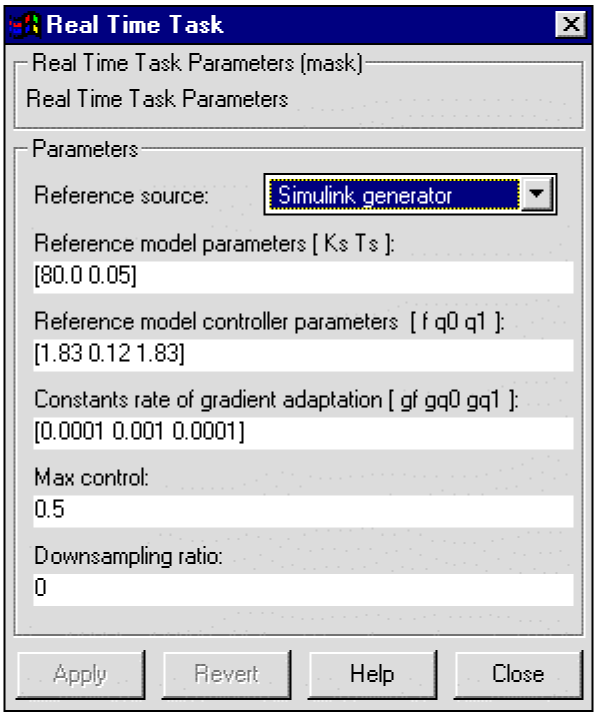


Figure 8-3: Simulink model for the *es_mrac.dll* library (*es_mracm.mdl* file).


Real Time Task

Real Time Task Parameters (mask)

Real Time Task Parameters

Parameters

Reference source: Simulink generator

Reference model parameters [Ks Ts]:
[80.0 0.05]

Reference model controller parameters [f q0 q1]:
[1.83 0.12 1.83]

Constants rate of gradient adaptation [gf gq0 gq1]:
[0.0001 0.001 0.0001]

Max control:
0.5

Downsampling ratio:
0

Figure 8-4: Parameters of the *Real-Time Task* block.

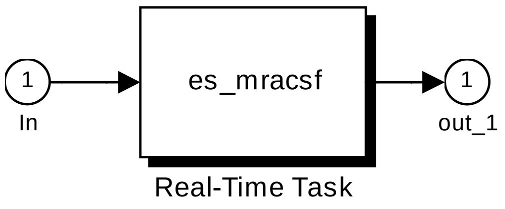
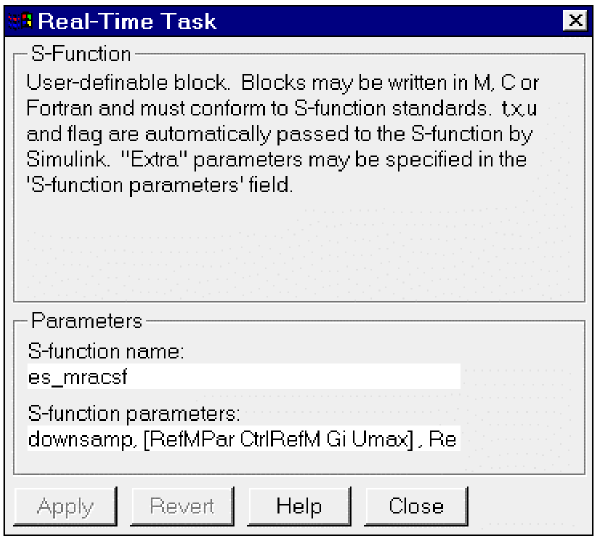


Figure 8-5: The *Real-Time Task* block.



User-definable block. Blocks may be written in M, C or Fortran and must conform to S-function standards. t,x,u and flag are automatically passed to the S-function by Simulink. "Extra" parameters may be specified in the 'S-function parameters' field.

S-function name:
es_mracsf

S-function parameters:
downsamp, [RefMPar CtrlRefM Gi Umax] , Re

Figure 8-6: Parameters of the *es_mracsf* S-function block.

Mask edit fields are:

Mask field	Parameter (s)
Reference model parameters [Ks Ts]	$[K_{DCm}\ T_{DCm}]$
Reference model controller parameters [f q0 q1]	$[f_m\ \ q_{0m}\ \ q_{1m}]$
Constant rate of gradient adaptation [gf gq0 gq1]	$[\bar{g}_{f_p}\ \ \bar{g}_{q_{0p}}\ \ \bar{g}_{q_{1p}}]$
Initial servo controller parameters [f q0 q1]	$[f_p(0)\ \ q_{0p}(0)\ \ q_{1p}(0)]$
Max control [Umax]	|Umax|
Downsampling ratio	see S-function description

External library function calls used to set or to get the parameters are:

SetParam

Sample time,

Ks	Ks parameter of the reference model,
Ts	Ts parameter of the reference model,
f	value of the reference model controller parameter,
q0	value of the reference model controller parameter,
q1	value of the reference model controller parameter,
gf	value of the constant rate of gradient adaptation,
gq0	value of the constant rate of gradient adaptation,
gq1	value of the constant rate of gradient adaptation,
Umax	maximum control value [0 ÷ 1],

GetParam

Sample time,

Ks	Ks parameter of the reference model,
Ts	Ts parameter of the reference model,
f	value of the reference model controller parameter,
q0	value of the reference model controller parameter,
q1	value of the reference model controller parameter,
gf	value of the constant rate of gradient adaptation,
gq0	value of the constant rate of gradient adaptation,
gq1	value of the constant rate of gradient adaptation,
Umax	maximum control value [0 ÷ 1],

SetOutCtrlParam

f value of the process controller parameter,
q0 value of the process controller parameter,
q1 value of the process controller parameter,

GetOutCtrlParam

f value of the process controller parameter,
q0 value of the process controller parameter,
q1 value of the process controller parameter,

GetRefModelState

Reference model position,
Reference model velocity.

To use them execute the following commands:

```
dummy = es_call( 'CallExtIPC', 'SetParam', [ Par ] );
OutCtrlParam = es_call( 'CallExtIPC', 'GetOutCtrlParam',[1]);
```

8.4. EXPERIMENT 1

The main goal is to follow the desired position set as square wave.

The magnetic brake is "on" and the algorithm starts with the following parameters:

Parameter (s)	Value
$[K_{DCm}\ T_{DCm}]$	[80.0 0.05]
$[f_m\ \ q_{0m}\ \ q_{1m}]$	[0.092 0.058 0.092]
$[\overline{g}_{f_p}\ \ \overline{g}_{q_{0p}}\ \ \overline{g}_{q_{1p}}]$	[0.00002 0.00001 0.00002]
$[f_p(0)\ \ q_{0p}(0)\ \ q_{1p}(0)]$	[0.1 0.08 0.1]
|Umax|	0.6
Downsampling ratio	0

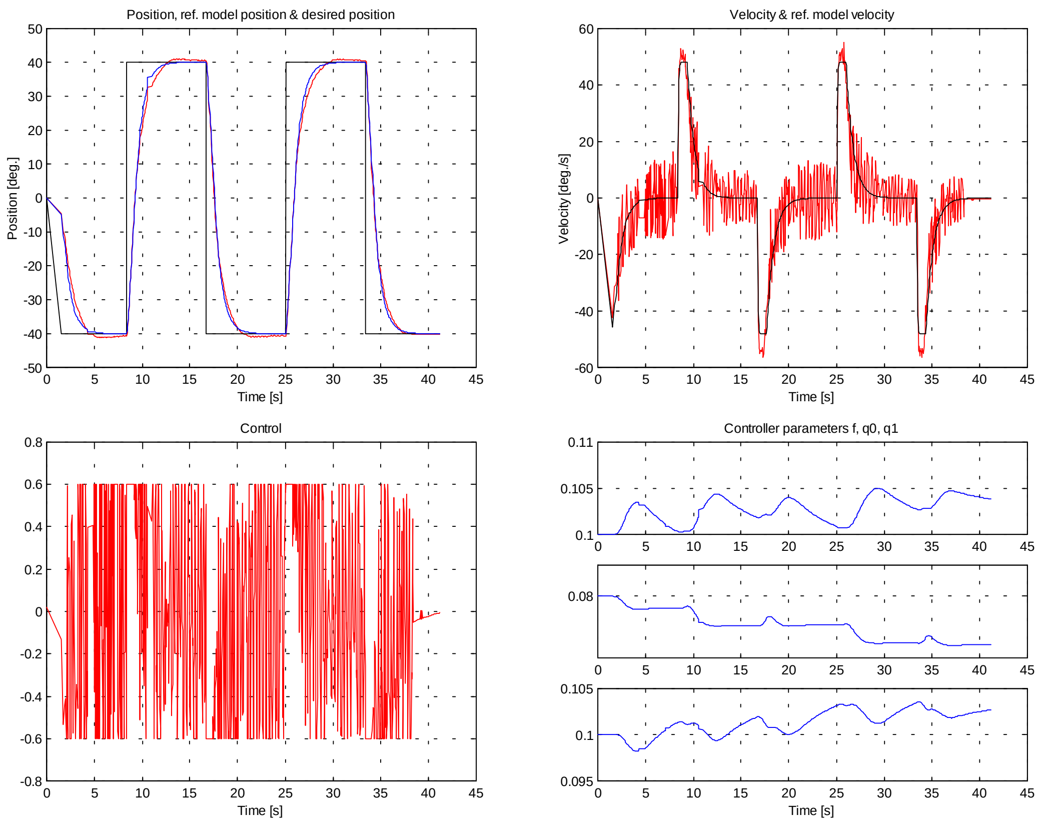


Figure 8-7: Tracking the reference signal (brake is „on")

Next, the brake is switched "off", and we observe the following differences in behaviour of the servo motor.

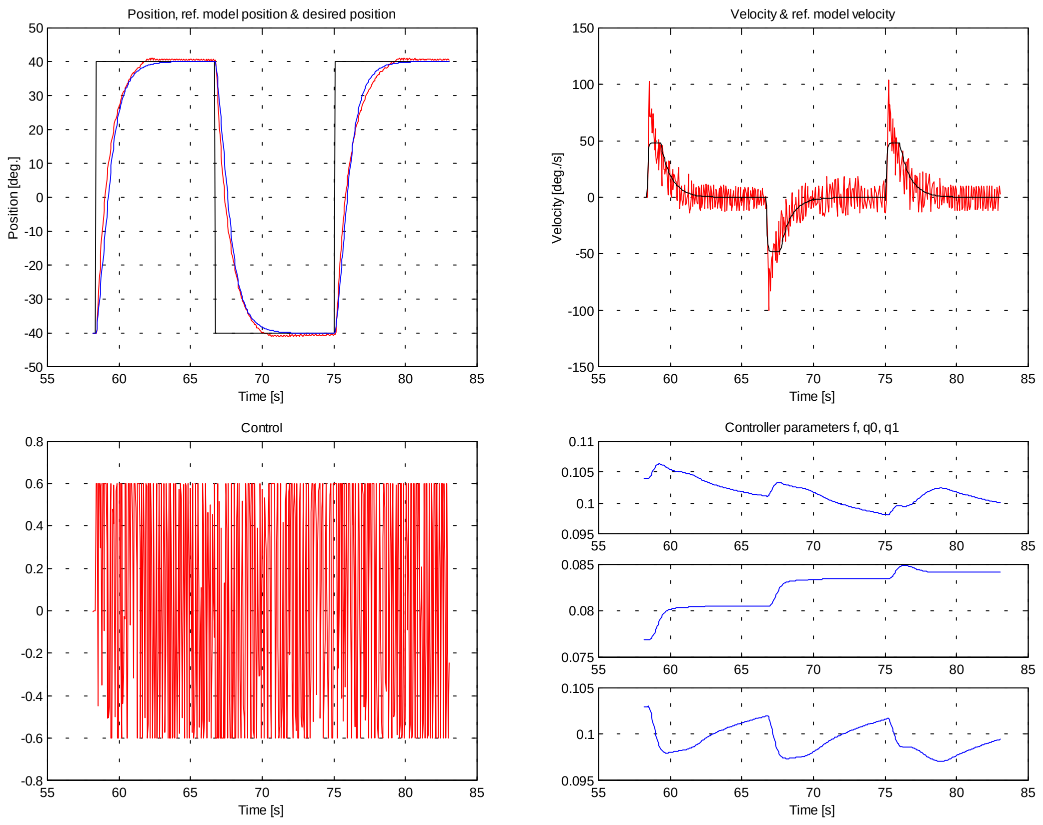
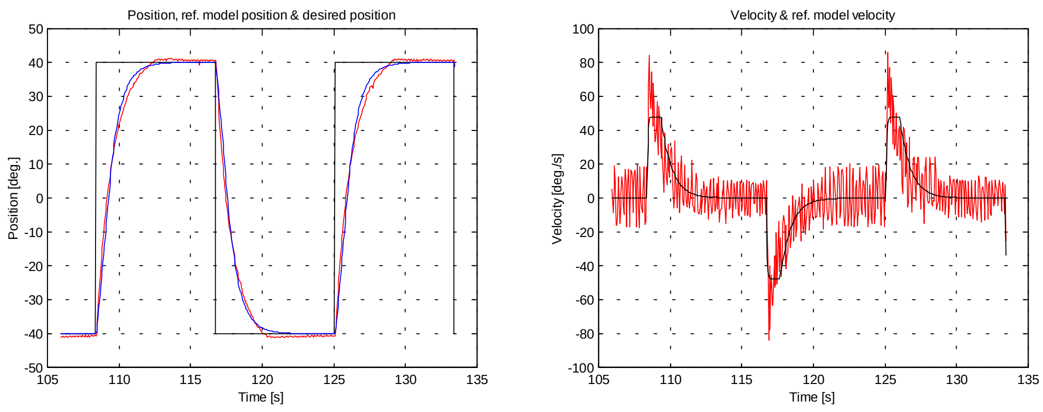


Figure 8-8: Tracking the reference signal before adaptation (brake is "off")

In this case we can observe that the parameters of the DC motor controller starts to change.

After few periods we can observe better adaptation of the servo-motor response to the reference model one.



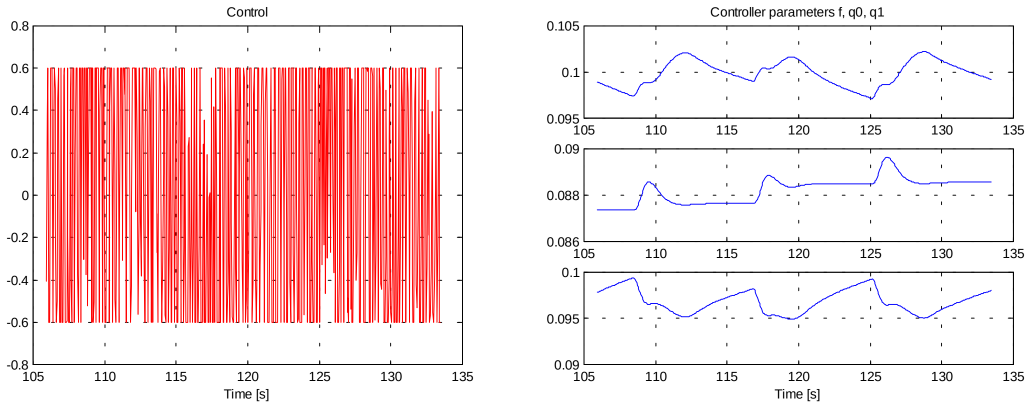


Figure 8-9: Tracking the reference signal after adaptation (brake is off)

8.5. C LANGUAGE IMPLEMENTATION

The source code of the external controller used to calculate adaptive control is shown below. Only the part responsible for the control value calculation is shown. To see the entire C-source code refere to the *es_mrac.c* file.

```
void __export __stdcall ExternalController(double far *p,
        double Chan1, double Chan2, double Chan3, double Chan4,
        double Chan5, double Chan6, double Time, double far *Control)
{
 //
 // Chan1 - position
 // Chan2 - velocity
 // Chan3 - from internal built-in signal generator
 //
 double     *pOut;
 double      FiltPosOut;
 double      FiltVelOut;
 double      DesValue=Chan3;
 double      DCPos=Chan1;
 double      DCVel=Chan2;
 double      Ctrl;
 double      OutCtrl;
 double      Error;
 double      dummy;

 //Model Calculation
 Eq2Calculate=0;
 Ctrl=fCtrl*DesValue-q1Ctrl*RefModelPos-q0Ctrl*RefModelVel;
 if(Ctrl>Umax) Ctrl=Umax;
 if(Ctrl<-Umax) Ctrl=-Umax;
 PrevRefModelPos=RefModelPos;
 output[0]=outputRefMod[0];  output[1]=outputRefMod[1];
 pOut=SingleStep(Ctrl);
 outputRefMod[0]=output[0];  outputRefMod[1]=output[1];
 RefModelPos=*(pOut+0);
 RefModelVel=*(pOut+1);

 //Filter Position calculation
 Eq2Calculate=1;
 output[0]=outputPosFilt[0];  output[1]=outputPosFilt[1];
 pOut=SingleStep(DCPos);
 outputPosFilt[0]=output[0];  outputPosFilt[1]=output[1];
 FiltPosOut=*(pOut+0);
 dummy=*(pOut+1);
```

```
    //Filter Velocity calculation
    Eq2Calculate=2;
    output[0]=outputVelFilt[0];  output[1]=outputVelFilt[1];
    pOut=SingleStep(DCVel);
    outputVelFilt[0]=output[0];  outputVelFilt[1]=output[1];
    FiltVelOut=*(pOut+0);
    dummy=*(pOut+1);

    //Controller coefficient calculation
    Error=DCPos-RefModelPos;
    OutfCtrl+=-SamplePeriod*Gf*Error*RefModelPos;
    Outq0Ctrl+=SamplePeriod*Gq0*Error*FiltVelOut;
    Outq1Ctrl+=SamplePeriod*Gq1*Error*FiltPosOut;
    OutCtrl=OutfCtrl*DesValue-Outq1Ctrl*DCPos-Outq0Ctrl*DCVel;
    PrevDCPos=DCPos;
    if(OutCtrl>Umax) OutCtrl=Umax;
    if(OutCtrl<-Umax) OutCtrl=-Umax;

    *Control = OutCtrl;
  }
```

Bibliography

[1] Iserman R., Lachman K., Matko D. : Adaptive Control Systems, Prentice Hall International 1992.

Notes

9 Example 4: Simulink Model for the External Controller

Simulink models may supervise the behaviour of the DC motor system and external controllers. There are two Simulink models available - one for the *es_sto* controller and the other for the *es_mrac* controller. The names of the appropriate Simulink models are *es_stom* (stored in the *es_stom.mdl* file) and *es_mracm* (the *es_mracm.mdl* file). The implementation is similar in both cases. This section describes the implementation of the *es_stom* Simulink model in details.

The Simulink model for monitoring of the experiments with the *es_sto.dll* library is shown in Figure 9-1. **The main function of this model is not to simulate the system but to transfer data to and from the Real Time Kernel.** This is the main difference between this model and a typical application of Simulink models. The Simulink model works in this case as an graphical front-end of the external DLL library.

The main block of the presented Simulink model is *Real Time Task*. It contains the S-function block calling the *es_stosf.m* S-function (see Figure 9-2). *Real Time Task* is exceeded by the signal generator. The signal generator may be the source of reference position of the servo disk. The output of the *Real Time Task* is a vector which contains information about angles of the DC motor shaft, velocity of the shaft, reference position of the shaft and control value. 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).

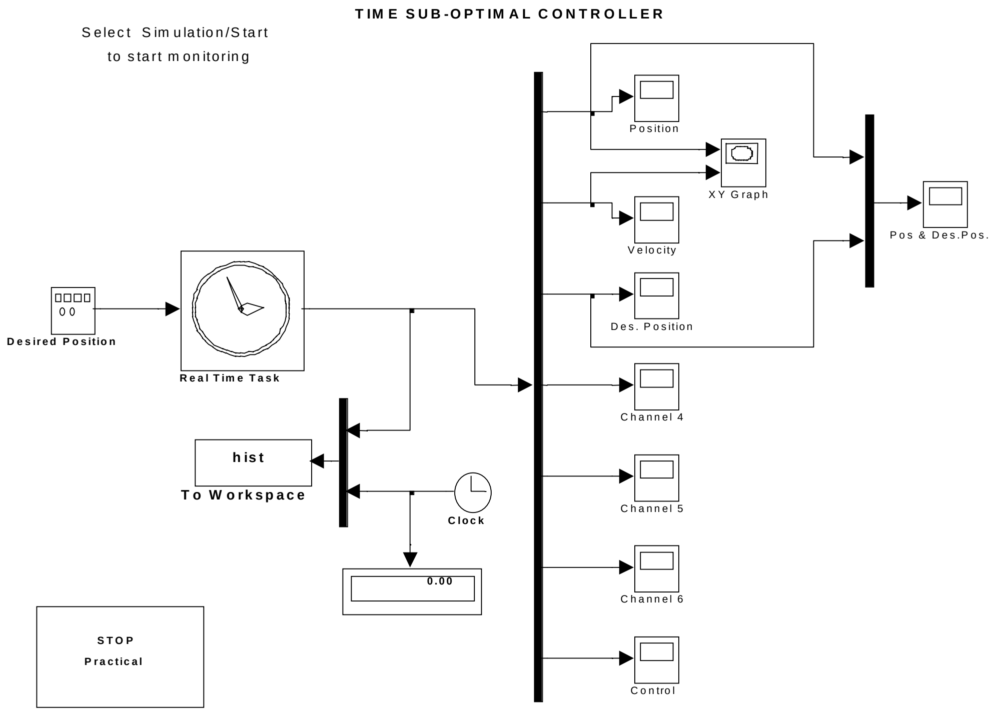


Figure 9-1: Simulink model for the *es_sto.dll* library (*es_stom.mdl* file).

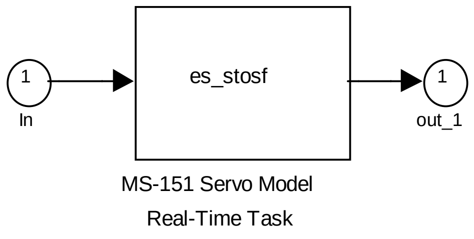


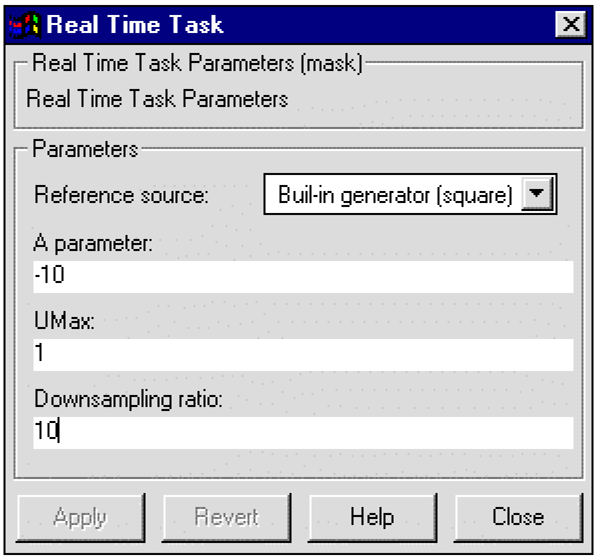
Figure 9-2: The *Real-Time Task* block.

The *es_stosf.m* file contains the S-function associated with the *Real Time Task* block. The function has four parameters:

- *Reference source* - this parameter is set by selecting the appropriate element in the listbox which contains three fields:
  - ◊ *Simulink generator* - this option connects the Simulink signal generator *Desired Position* (see Figure 9-1) as the source of the reference angle,
  - ◊ *External signal* - the reference signal comes from the MS151hardware,

◊ *Built-in generator (square)* - the signal generator built into the RTK is the source of the square wave which determines reference value,

- the *A parameter* - determines the angle of the switching line,
- *Umax* - maximum value of the control in the range from –1.0 to +1.0,
- *Downsampling ratio* - this parameter decreases the rate of output data flow from the output of the *Real Time Task* (see Figure 9-3). For example, if the downsampling coefficient is 10, for each 10 sampling periods only one is transfered to the output of the block.

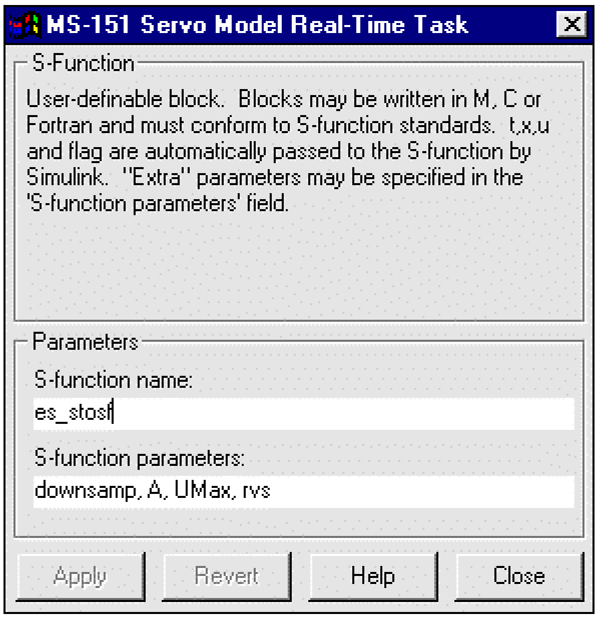

Real Time Task

Real Time Task Parameters (mask)
Real Time Task Parameters

Parameters
Reference source: Buil-in generator (square)
A parameter:
-10
UMax:
1
Downsampling ratio:
10

Figure 9-3: Parameters of the *Real-Time Task* block.

The S-function block included in the *Real Time Task* block contains the S-function file name (*es_stosf*) and four parameters: the downsampling coefficient value (*downsamp*), the A parameter (*A* variable), the UMax value (*UMax* variable) and the value which determines the reference value source (*rvs* variable) (see Figure 9-4).


MS-151 Servo Model Real-Time Task

S-Function

User-definable block. Blocks may be written in M, C or Fortran and must conform to S-function standards. t,x,u and flag are automatically passed to the S-function by Simulink. "Extra" parameters may be specified in the 'S-function parameters' field.

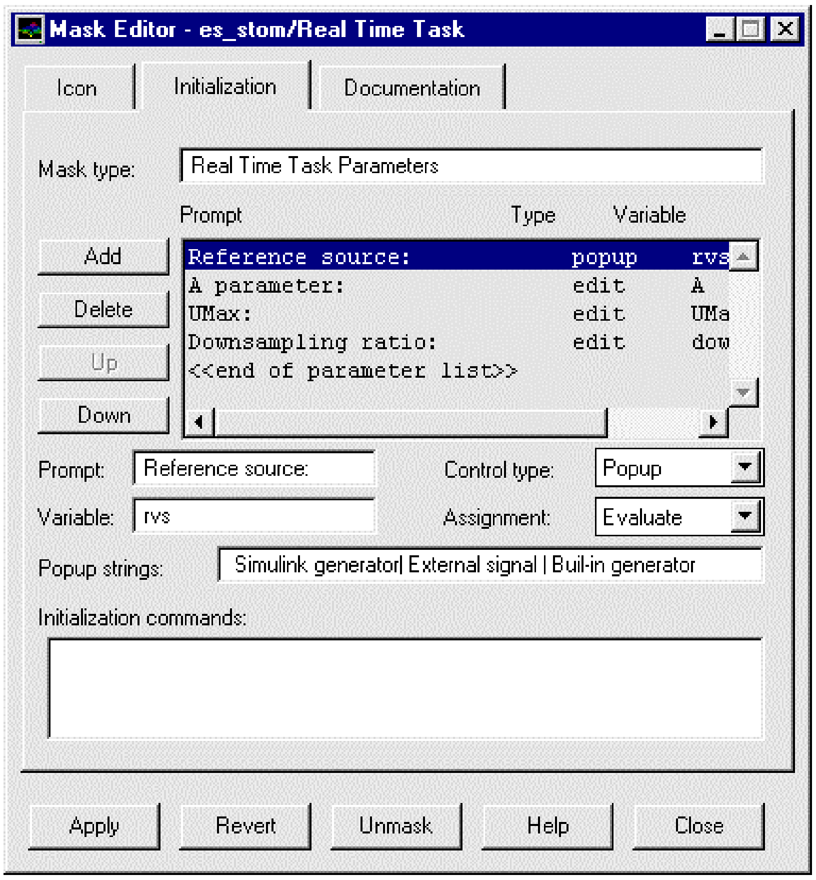
Parameters

S-function name:
es_stosf

S-function parameters:
downsamp, A, UMax, rvs

Figure 9-4: Parameters of the *es_stosf* S-function block.

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.



Mask type: Real Time Task Parameters

```
Prompt                                     Type      Variable
Reference source:                          popup     rvs
A parameter:                               edit      A
UMax:                                      edit      UMa
Downsampling ratio:                        edit      dow
<<end of parameter list>>
```

Prompt: Reference source: Control type: Popup

Variable: rvs   Assignment: Evaluate

Popup strings: Simulink generator| External signal | Buil-in generator

Initialization commands:

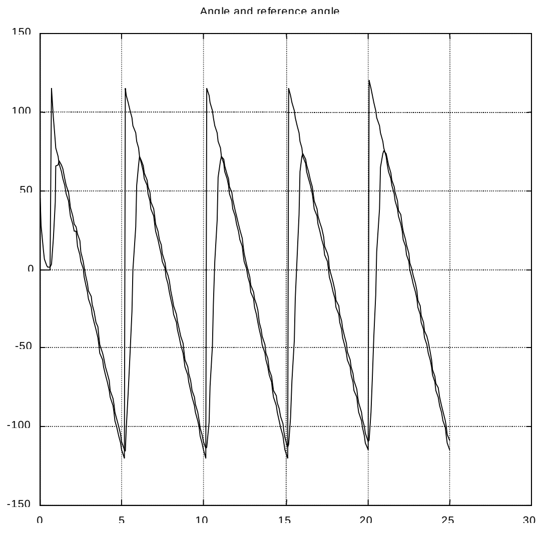
Figure 9-5: Mask definition for the *Real-Time Task* block.

The block *STOP Practical* block (Figure 9-1) is used to switch off control for the DC motor. Double clicking the mouse over this block executes the following command:

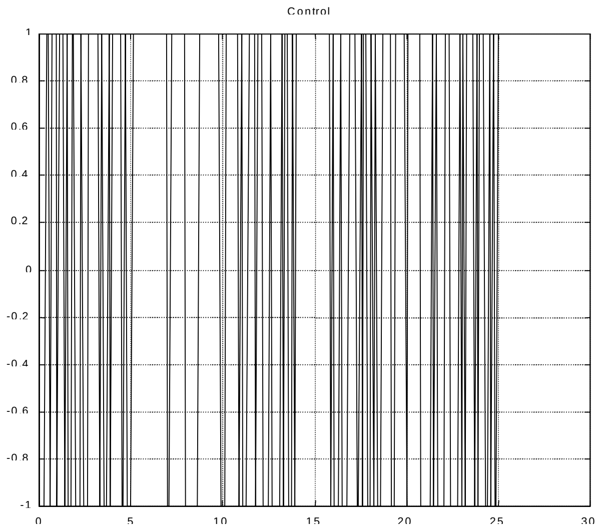
es_call('StopPractical')

which sets control values to zero.

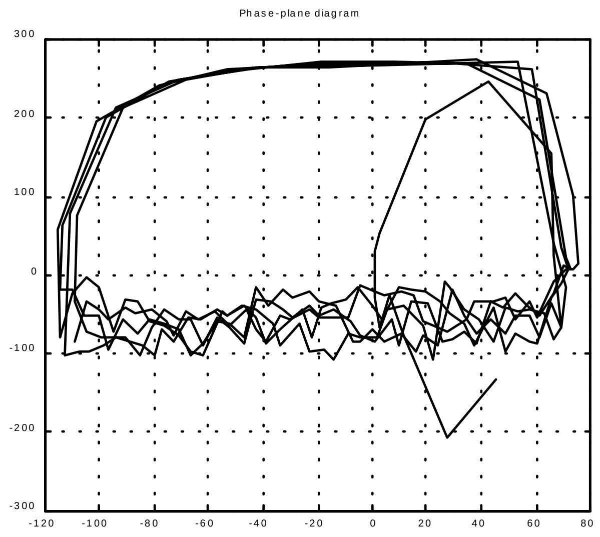
The history of the experiment is transferred to the MATLAB workspace as the *hist* experiment. The statements given below can be used to display the history of the experiment. The sawtooth signal from the Simulink signal generator was used as the reference value *plot(hist(:, 8), hist(:, [1 3])); grid; title('Angle and reference angle')*



plot(hist(:, 8), hist(:, 7)); grid; title('Control')



plot(hist(:, 1), hist(:, 2)); grid; title('Phase-plane diagram')



The "body" of the *es_stosf.m* S-function file is shown below. Notice that the S-function uses the *flag* input argument in an untypical way. 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 angle.

The value of reference position can be taken from the signal generator, from an external source or calculated by the built-in signal generator (Fig.9.1) .The reference position from the Simulink signal generator is passed to the S-function as the input argument *u*.

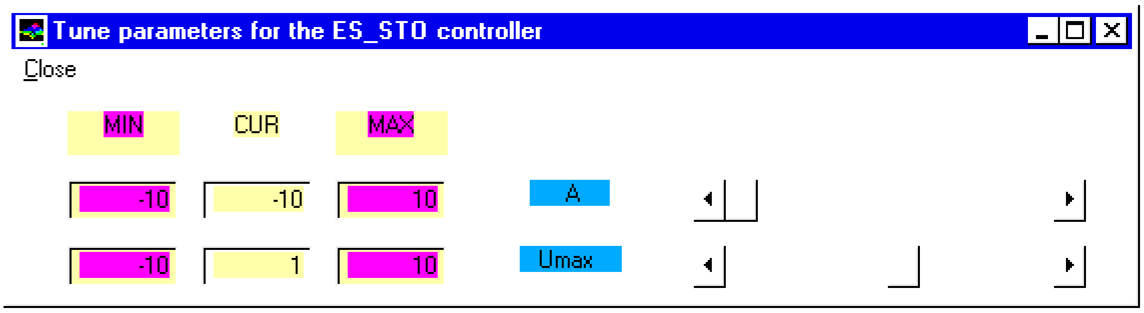
The flag equal to 3 is used to read the history of the experiment from the Real-Time Kernel and to transfer it to the output of the *Real Time Task* block. The flag equal to 4 is used to calculate the next discrete time point. The functions associated with the flags 3 and 4 call the *es_sfco* and *es_sfntp* functions which are 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, A, UMax, rvs )

% Global variables
global 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, 7 outputs, 1 inputs
    sizes.NumContStates  = 0;
    sizes.NumDiscStates  = 1;
    sizes.NumOutputs     = 7;
    sizes.NumInputs      = 1;
    sizes.DirFeedthrough = 0;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);

    % Set initial values of the control algorithm
    dummy = es_call( 'SetAlgNo', 0 );

    T0 = 0.01;
    dummy = es_call( 'SetSampleTime', T0 );
    dummy = es_call( 'SetInitCond', [ 0 0 ] );
    switch rvs,
      case 1, % Simulink signal generator
        dummy = es_call( 'SetDataSource', [ 8 9 13 4 5 6 ] );
        dummy = es_call( 'SetPW', [ 0 0 zeros( 1, 18 ) ] );
      case 2, % External signal source
        dummy = es_call( 'SetDataSource', [ 8 9 12 4 5 6 ] );
      case 3, % Built-in signal generator (square)
        dummy = es_call( 'SetDataSource', [ 8 9 13 4 5 6 ] );
        dummy = es_call( 'SetPW', ...
                    [ 1 3 3 3 3 -50 -50 50 50 zeros( 1, 11 ) ] );
    end
    dummy = es_call( 'LoadExtAlg', 'es_sto.dll' );
    dummy = es_call( 'CallExtIPC', 'SetParam', [ A UMax ] );
    dummy = es_call( 'SetAlgNo', 99 );  % external controller
    dummy = es_call( 'ResetTime' );


    % Wait for the first sample which may be send to the output
    while ( es_call( 'GetNoOfSamples' ) <= downsamp )
                ;
    end;
    history=es_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
    if rvs == 1  % Simulink signal generator
      dummy = es_call( 'SetPW', [ 0 u zeros( 1, 18 ) ] );
    end
    sys = x;

case 3, % Calculate outputs
   [ sys, downsamp, history ] = es_sfco( downsamp, history );

case 4,% Calculate next discrete time point
   [ sys, downsamp, history ] = es_sfntp( downsamp, history );

case 9, % Termination
    dummy = es_call( 'SetAlgNo', 0 );
    dummy = es_call( 'UnloadExtAlg' );

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

There is available a GUI interface which may be used for *on-line* parameter tuning of the *es_sto.dll* lqbrary. The interface window is called by the *es_stosl* command. The GUI window contains the edit fields and sliders, which can set minimum, maximum and current value of all parameters (see Figure 9-6).

See the body of the *es_stosl.m* file for implementation details of the GUI interface.



MIN	CUR	MAX	
-10	-10	10	A
-10	1	10	Umax

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

The presented GUI window appears only if the *es_sto.dll* library is loaded.

It is recommended to start simulation of the *es_stom* Simulink model first and next execute the *es_stosl* command. It causes the Simulink model to set new value of the reference angle and gives the ability to change parameters simultaneously.

10 QUICK REFERENCE TABLE

File Name (included in the EXTERNAL directory)	**Description**
mk_bcc.bat	batch file for the Borland C/C++ v.5.0 compiler
mk_msvc.bat	batch file for the Microsoft Visual C/C++ v.2.0 compiler
mk_wcc.bat	batch file for the Watcom C/C++ v.10.6 compiler
es_sto.c	example of the C-source code of the external DLL library
es_mrac.c	example of the C-source code of the external DLL library
es_tmf.c	template C-source code of the external DLL library
es_sto.dll	compiled version of the *es_sto.c* file
es_mrac.dll	compiled version of the *es_mrac.c* file
es_stom.mdl	Simulink model for the *es_sto.dll* DLL library
es_mracm.mdl	Simulink model for the *es_mrac.dll* DLL library
es_stosf.m	S-function for the *es_stom.mdl* Simulink model
es_mracsf.m	S-function for the *es_mracm.mdl* Simulink model
es_stosl.m	GUI interface for the *es_sto.dll* executable

Notes