IMPLEMENTATION OF SLIDING MODE CONTROL IN SIMULINK 4.1 (R12.1)

Květoslav Belda
Institute of Information Theory and Automation
Academy of Sciences of the Czech Republic

Abstract: The paper briefly introduces Sliding Mode Control (SMC) of the planar redundant parallel robot. This type of the robots seems to be one of the promising ways to solve the problems of accuracy and speed. The main goal is presentation of using MATLAB – SIMULINK environment version 4.1 (R12.1) for simulation of such control of the robot. Implementation is accomplished by combination of S-Functions and ordinary Functions, both programmed in C code as mex Functions.

1. Introduction

The area of the robots and manipulators is in the constant development caused by the fact that the robots are the basis of the most machine and production lines in the factories.

The uncompromising requirements on their new types are primarily high accuracy, high speed. One example of the promising parallel construction is in Fig. 1. Figure shows the planar redundant parallel robot for which the Sliding Mode Control was designed.

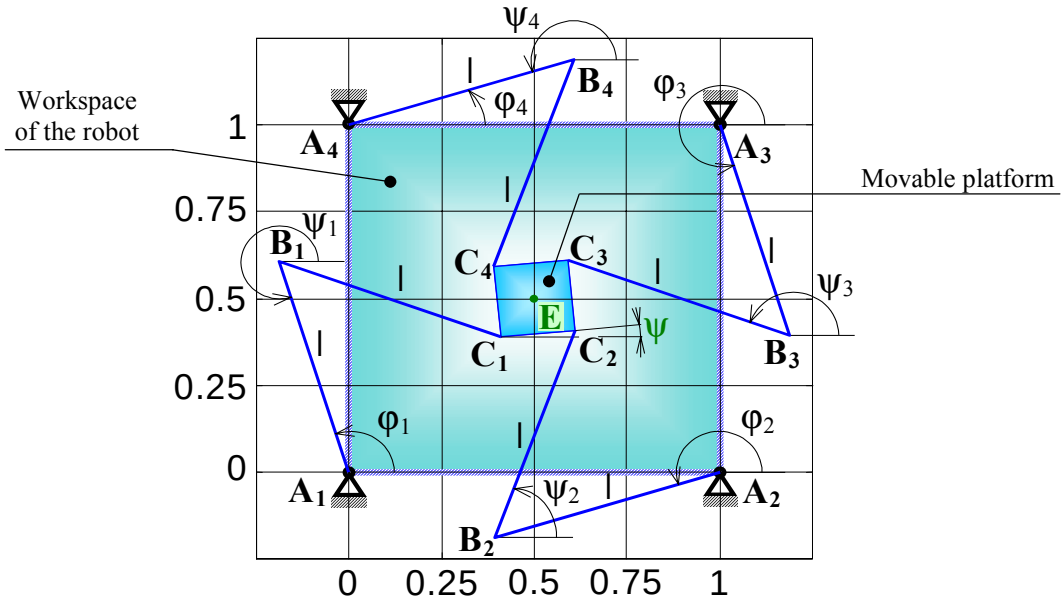


Fig. 1. Scheme of planar parallel robot with the most important geometrical description (the Cartesian coordinates (x_E, y_E, ψ) of center of movable platform, and all angles: motor angles ($\varphi_{1\text{-}4}$) and joint angles ($\psi_{1\text{-}4}$)).

Mechanical system (robot-manipulator Fig. 1) can be described by nonlinear differential equation in the following form.

$$\ddot{\mathbf{y}} = -\mathbf{f}\left(\mathbf{y}, \dot{\mathbf{y}}\right) + \mathbf{B}(\mathbf{y})\,\mathbf{u} \tag{1}$$

The equation we can obtain with using Lagrange's equations of mixed type. After this description of the robot we can start with introduction of the approach to the control.

2. Sliding Mode Control (SMC)

Discrete type of the Sliding mode control (Elmali 1992) is derived analogically to the theory of stability in continuous domain. Generally it is based on the 'switching' control action and the performance of Lyapunov stability theorem.

The state is driven towards a desired switching (sliding) hyperplane under Lyapunov control. The 'switching' maintains the state on this hyperplane, once it has been reached, in spite of perturbations. This method offers an advantage of accuracy at the cost of control dithering, which ensues from the 'switching' part.

Let us consider the nonlinear equation (1), which can be transformed and simply discretized by Taylor series with sampling period δ to the following state formulation:

$$\mathbf{X}(k+1) = \mathbf{A}(\mathbf{X}(k)) + \mathbf{B}(\mathbf{X}(k))\,\mathbf{u}(k) \tag{2}$$

With this state description, we can obtain control law in the following structure:

$$\mathbf{u}(k) = -(\mathbf{CB}(k))^{-1}\{\mathbf{C}[\mathbf{A}(k) + \Psi(k) - \mathbf{X}_d(k+1)] - \mathbf{s}(k+1)\}$$

$$\mathbf{u}(k) = \quad \widetilde{\mathbf{B}}^{-1} \quad (\mathbf{F}(\mathbf{X}, \mathbf{X}_d)) \tag{3}$$

where $\mathbf{s}(k+1) = e^{-P\delta}\mathbf{s}(k) - \mathbf{K}sign(\mathbf{s}(k))$ (4)

with considering $s(k) = \mathbf{C}(X(k) - Xd(k))$ (5)

which represents the choice of hyperplane (elements of matrix **C**) and where the evaluation of control error is included.

Eq. (4) with eq. (5) must satisfy Lyapunov stability theorem. $\mathbf{\Psi}(k)$ represents unknown perturbation, which can be estimated by

$$\mathbf{\Psi}(k-1) = \mathbf{X}_{\substack{actual \\ topical}}(k) - \mathbf{A}(k-1) - \mathbf{B}(k-1)\mathbf{u}(k-1) \tag{6}$$

With the assumption that the dynamics of perturbation is considerably slower than discretization frequency and the order of the perturbation magnitude is much smaller, the estimation is valid.

By this the Sliding Mode Control is briefly defined and now we can discuss its implementation in SIMULINK.

2. The implementation in SIMULINK environment.

For simulation of the robot, the two main parts are needed. The first part is a model of the robot, which is represented by mex S-Function SmodRob. This function provides computation of derivatives and state of the robot. For these, the S-Function calls the ordinary mex Function MclSMC.dll, which computes model of the robot. It means, it computes all elements of differential equation (1).

The second part is control part, which is represented by mex S-Function SmodSMC. This function uses actual topical state of the robot and desired values of this state (input). The SMC is based on the model, therefore S-Function computes model of the robot (elements of eq.(2); mex Function MclSMC.dll) and consecutively computes four actuators for four drives (mex Function clSMC.dll; output). The following subsections progressively introduce the SIMULINK block diagram, structures of functions and alert on interesting places in the code. The notes are appended into structure of the mex files.

2.1 The Simulink block diagram.

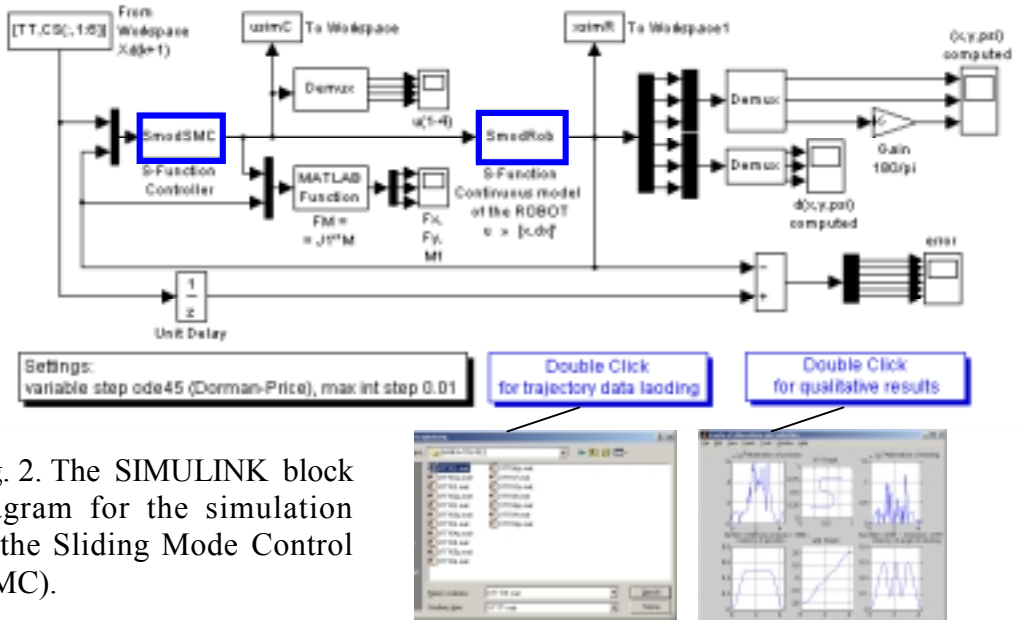


Fig. 2. The SIMULINK block diagram for the simulation of the Sliding Mode Control (SMC).

In figure, the mex S-Functions are boldly surrounded. For simple utilization, the diagram includes references on Graphical User Interface (GUI, buttons Double Click ···).

2.2 The structure of ordinary Functions in C code (mex files).

Executive part of the function.
First row includes outputs and rest is input.

```
#include <math.h>
#include "mex.h"
void Cfun_MclSMC(real_T f[3],real_T B[3*4],real_T Asmc[6],real_T Bsmc[6*4],
                 real_T X[6],
                 real_T Ts, real_T L, real_T l, real_T a, real_T m3)
{
/* Declaration of local (internal) variables ************************* */
 double m10, IT10, pom;  /* scalar variables */
 double m[4],I[4],r[4]  /* array variables  */
. . .
/* body of the Function ********************************************** */
. . .
}
```

MATLAB interface part of the function.

```
void mexFunction(int no, mxArray **out, int ni, const mxArray **in)
{
 if (ni != 6) { mexErrMsgTxt("6 input arguments required."); return;}
 if (no != 4) { mexErrMsgTxt("4 output arguments required."); return;}

 out[0] = mxCreateDoubleMatrix(3,1,mxREAL); /* outputs: continuous model */
 out[1] = mxCreateDoubleMatrix(3,4,mxREAL); /* outputs: continuous model */
 out[2] = mxCreateDoubleMatrix(6,1,mxREAL); /* outputs: discrete model */
 out[3] = mxCreateDoubleMatrix(6,4,mxREAL); /* outputs: discrete model */
```

Declaration and definition of outputs.

Call of executive part of the function.
First row includes outputs and rest is input.

```
/* Cfun_MclSMC(f, B, Asmc, Bsmc,
                X,
                Ts, L, l,
                a, m3)
Cfun_MclSMC(mxGetPr(out[0]),mxGetPr(out[1]),mxGetPr(out[2]),mxGetPr(out[3]),
             mxGetPr(in[0]),
            *mxGetPr(in[1]),*mxGetPr(in[2]),*mxGetPr(in[3]),
            *mxGetPr(in[4]),*mxGetPr(in[5]));
}
```

2.3 The S-Functions in C code (mex files).

Example of structure of the S-Function SmodRob for computation of the continuous model of the robot.

```
#define S_FUNCTION_NAME SmodRob
#define S_FUNCTION_LEVEL 2
#include <math.h>
#include <simstruc.h>
/* Define for easy access of the input parameters
                                  sGetSFcnParam(SimStruct *S, int_T index) */
#define mxpa_X0 ssGetSFcnParam(S,0)      /* pointer to initial condition */
#define mxpa_Ts ssGetSFcnParam(S,1)            /* pointer to sample time */
. . .
#define u1(element) (*u1Ptrs[element])         /* pointer to Input Port0 */
. . .
. . .
/* mdlInitializeSizes */
static void mdlInitializeSizes(SimStruct *S) { body of the function }

/* mdlInitializeSampleTimes */
static void mdlInitializeSampleTimes(SimStruct *S) { body of the function }

/* mdlInitializeConditions */
#define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions( SimStruct *S) { body of the function }

/* mdlOutputs */
static void mdlOutputs(SimStruct *S, int_T tid) { body of the function }

/* mdlUpdate */
#define MDL_UPDATE
static void mdlUpdate(SimStruct *S, int_T tid) { body of the function }

/* mdlDerivatives */
#define MDL_DERIVATIVES
static void mdlDerivatives(SimStruct *S) { body of the function }

/* mdlTerminate */
static void mdlTerminate(SimStruct *S) { }

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"                    /* MEX-File interface mechanism */
#else
#include "cg_sfun.h"           /* Code generation registration function */
#endif
```

Macro for getting of inputs.

Inside of the S-Function, where the ordinary Functions (mex) are called.

```
InputRealPtrsType u1Ptrs=ssGetInputPortRealSignalPtrs(S,0);
real_T *xcont  = ssGetContStates(S);
real_T *dxcont = ssGetdX(S);
double pom, *f, *B; mxArray *lhs[4], *rhs[6];
int i, j;
{
 rhs[0] = mxCreateDoubleMatrix(6, 1, mxREAL); /* rhs=right hand side par. */
 for (i=0;i<6;i++) mxGetPr(rhs[0])[i]=xcont[i];
 rhs[1] = mxpa_Ts; rhs[2] = mxpa_L; rhs[3] = mxpa_l; rhs[4] = mxpa_a;
 rhs[5] = mxpa_m3;
 mexCallMATLAB(4, lhs, 6, rhs, "MclSMC");
 f=mxGetPr(lhs[0]); B=mxGetPr(lhs[1]); /* lhs=left hand side (parameters) */
 . . .
 for (j=0; j< 4; j++) pom+=B[j*3+i]*u1(j);
     dxcont[i+3]=f[i]+pom;    . . .
}
```

Getting of inputs.

mexCallMATLAB of fun MclSMC.dll

Transfer of outputs of the mex Function MclSMC.dll to variables.

Use of the macro for getting of inputs.

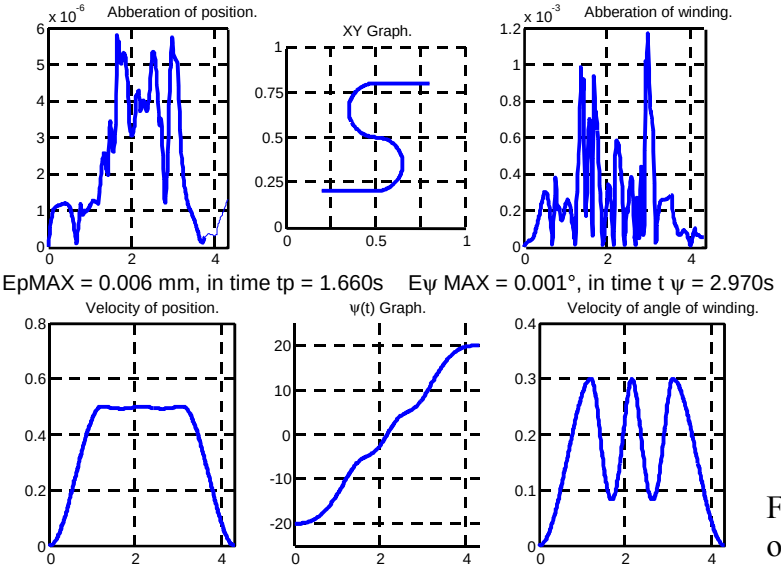
Note: Simple example of handling with matrix in format, which is compatible with MATLAB saving procedure of arrays (matrixes). The example uses the mexPrintf command to list elements of marix. Matlab command line gives back for:

```
>> B=[1 2 3; 4 5 6; 7 8 9]
   B =
        1     2     3
        4     5     6
        7     8     9
```

in Matlab the matrix-arrays are saved like this: B = [1 4 7 2 5 8 3 6 9], for handling with such a m × n matrix B is suitable use versatile notation (one dimension arrays), which can be used in both cases: for Matlab even for ANSI norm of C code.

```
for (i=0; i < m; i++)
    {
     mexPrintf(" \n");
     for (j=0; j < n; j++) mexPrintf("%8.2f  \t", Bkm1[j*m+i]);
    }
result:     1.00     2.00     3.00
            4.00     5.00     6.00
            7.00     8.00     9.00
```

3. Illustration of simulation.



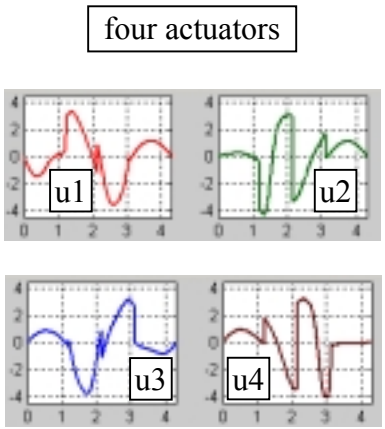


Fig. 3. Simulation of the tracking of the robot along the S_{shape} trajectory.

4. Conclusion.

The paper deals with C code implementation of Sliding Mode Control into Simulink environment. The programming in langue C was appeared as necessity for future real-time control of the physical model of the robot by DSP.

References

1. Online Manuals (in PDF): External Interfaces; Writing S-Functions. The MathWorks, Inc. 2000.
2. ELMALI, H. – OLGAC, N.: Sliding mode control with perturbation estimation (SMCPE): a new approach, INT. J. CONTROL 1992, vol.56, no.4, pp. 923 – 941.
3. BÖHM, J. – BELDA, K. – VALÁŠEK, M.: Study of control of planar redundant parallel robot. Proceedings of the IASTED International conference MIC 2001, pp. 694 – 699.

Acknowledgement

This research is supported by GAČR (101/99/0729, 1999-2001): "Redundant parallel robot and its control" and IG ČVUT (CTU IG 300104412, 2001) "The Direct kinematics for Parallel Robots".

Contact address:

Ústav teorie informace a automatizace AV ČR
Pod vodárenskou věží 4, 182 08 Praha 8 – Libeň
E-mail: belda@utia.cas.cz