REAL-TIME SIMULATION AND VISUALIZATION OF HIGH-LEVEL MODEL-BASED CONTROL OF REDUNDANT PARALLEL ROBOTS

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

Abstract: The paper briefly introduces examples of high-level model-based control (Sliding Mode Control – SMC and Generalized Predictive Control – GPC) applied to the redundant parallel robots. The objective is a presentation of using MATLAB – SIMULINK environment for real-time simulation and visualization of control of these robots. Implementation is accomplished by combination of C MEX S-Functions and ordinary C MEX-Files within SIMULINK scheme under support of Real-Time Windows Target (2.1) and Virtual Reality Toolbox (2.0).

1. Introduction

Topically, the next development in industrial area is constrained by deficit of powerful machines with proportional dynamics and stiffness. Utilization of parallel robots, combined with high-level model-based control, seems to be promising way, which solve this problem i.e. the problem of dynamics, stiffness, accuracy and productivity.

Scheme of one exemplary prototype of the parallel structures is in Fig. 1. Figure shows essential components of the planar redundant parallel robot.

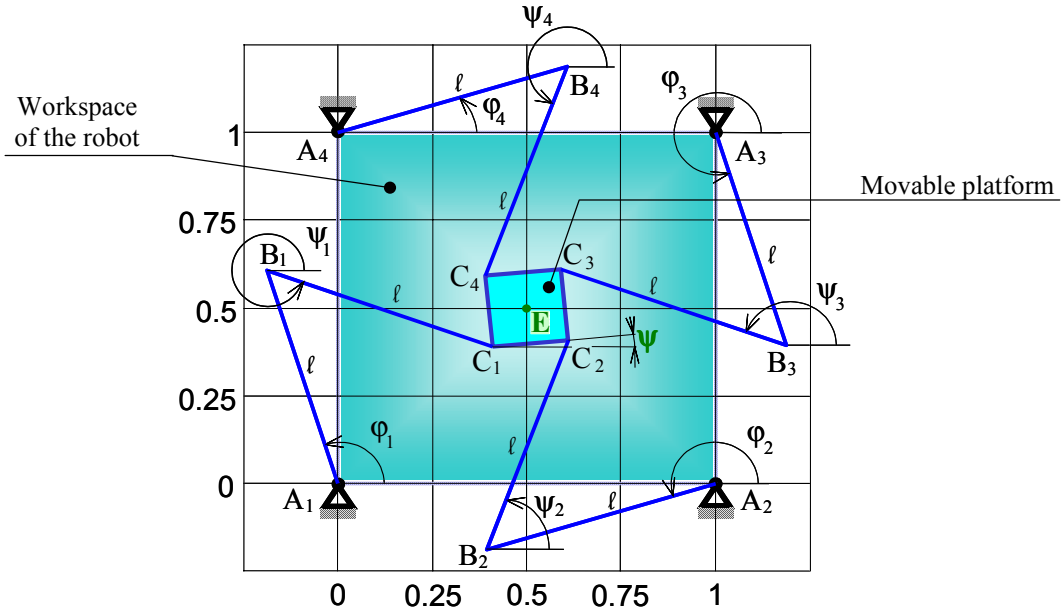

Fig. 1. Scheme of planar parallel robot with the most important geometrical description (the Cartesian coordinates (x_E, y_E, ψ) of center of movable platform, and all angles: motor angles ($\varphi_{1\text{-}4}$) and joint angles ($\psi_{1\text{-}4}$)).

The objective of this paper is a presentation of using MATLAB – SIMULINK environment for real-time simulation and visualization of high-level model-based control of mentioned robots, represented by robot in Fig. 1. Implementation is accomplished by combination of C MEX S-Functions and ordinary C MEX-Files within SIMULINK scheme under support of Real-Time Windows Target (2.1) and Virtual Reality Toolbox (2.0).

2. High-level model-based control

From the control point of view, the principal task of a robot is its movement along a planned trajectory. It is usually given in Cartesian coordinates, which are easily comprehensible for users. In these coordinates, the robot motion can be described by the following system of nonlinear differential equations [2, 4]:

$$\mathbf{R}^T\mathbf{M}\mathbf{R}\ddot{\mathbf{y}} + \mathbf{R}^T\mathbf{M}\dot{\mathbf{R}}\dot{\mathbf{y}} = \mathbf{R}^T\mathbf{g} + \mathbf{R}^T\mathbf{T}\mathbf{u} \tag{1}$$

where **M** is a mass matrix, **g** is a vector of right sides, **T** is a redistributional matrix, **R** is a Jacobian matrix, **y** is a robot output (Cartesian coordinates of movable platform, $\mathbf{y} = [x_E, y_E, \psi]$) and **u** is an input vector. This model (system of differential equations (1)) can be rewritten to different forms according to the requirements of appropriate control approach. Since, the robot represents multibody system with relatively time-consuming computation of its model e.g. (1), the choice of control strategies focuses on discrete methods. They can simply respect requirements to the computation time.

Now, after formulation of the robot model, we can continue with brief introduction of high-level model-based control approaches: firstly, of Sliding Mode Control (SMC) and consecutively, of Generalized Predictive Control (GPC).

2.1 Sliding Mode Control (SMC)

Discrete type of Sliding Mode Control [3] is derived in analogy of the theory of stability in continuous domain. In general, it is based on 'switching' action and accomplishment of Lyapunov theorem of the stability.

Let us consider system of differential equations (1), which can be simply transformed and discretized by Taylor series with sampling δ in the following state description:

$$\mathbf{X}(k+1) = \mathbf{A}\big(\mathbf{X}(k)\big) + \mathbf{B}\big(\mathbf{X}(k)\big)\,\mathbf{u}(k) \quad (\mathbf{X} = [\mathbf{y}, \dot{\mathbf{y}}] = [x_E, y_E, \psi, \dot{x}_E, \dot{y}_E, \dot{\psi}]) \tag{2}$$

If we use this description (2), we can obtain control law in the following structure [3]:

$$\mathbf{u}(k) = -\big(\mathbf{C}\mathbf{B}(k)\big)^{-1}\big\{\mathbf{C}\big[\mathbf{A}(k) + \boldsymbol{\Psi}(k) - \mathbf{X}_d(k+1)\big] - \mathbf{s}(k+1)\big\} \tag{3}$$

where $\mathbf{s}(k+1) = e^{-P\delta}\mathbf{s}(k) - \mathbf{K}\,sign(\mathbf{s}(k))$, on the assumption that $\mathbf{s}(k) = \boldsymbol{f}(\mathbf{X}(k) - \mathbf{X}_d(k))$, represents the choice of sliding hyppersurface, matrix **C** is a choice of the sliding mode and vector **Ψ** represents difference of the robot model from real measurement.

2.2 Generalized Predictive Control (GPC)

Predictive Control [3] is a multi-step control based on local optimization of quadratic criterion (4) with linearized discrete state description (5) (obtained on the base of [5]):

$$J_k = \boldsymbol{\mathcal{E}}\left\{ (\hat{\mathbf{y}} - \mathbf{w})^T(\hat{\mathbf{y}} - \mathbf{w}) + \mathbf{u}^T\boldsymbol{\lambda}\,\mathbf{u} \right\} \tag{4}$$

$$\begin{aligned} \mathbf{X}(k+1) &= \mathbf{A}\,\mathbf{X}(k) + \mathbf{B}\,\mathbf{u}(k) \\ \mathbf{y}(k) &= \mathbf{C}\,\mathbf{X}(k) \end{aligned} \quad \rightarrow \quad \hat{\mathbf{y}} = \mathbf{G}\mathbf{u} + \mathbf{f} \text{ (prediction of future outputs)} \tag{5}$$

If we make minimization of the quadratic criterion in the root form (i.e. $J_k = \mathbf{J}^T\mathbf{J}$) [4], we obtain control law for force effects **FM**, acting directly to the movable platform. We recompute them back (eq. 6) to real control actions of drives **u** according to equations (1):

$$\mathbf{u}(k) = \mathbf{F}\mathbf{M} \quad \rightarrow \quad \mathbf{R}^T\mathbf{T}\,\mathbf{u}_{\text{on drives}} = \mathbf{F}\mathbf{M} \tag{6}$$

Now, the both controls can be already used for real-time purpose.

3. Executive control procedures

This chapter introduces implementation of control algorithms trough capsulation of their C code in a form of S-functions into SIMULINK blocks and alternative using of basic blocks as structural elements for auxiliary tasks (loading of desired trajectory, for direct kinematic transformation etc.).

Essential structure of controllers of sliding and predictive algorithms - SMC and GPC is generally characterized as follows: First part is composed of computing of robot model, provided by functions MclSMC.c for SMC and MclGPC.c for GPC. The second part is composed of computations of real controllers by functions clSMC.c and clGPC.c. Both these parts are consecutively connected within S-functions (SmodSMC.c and SfunGPC.c), which are capsulated in SIMULINK blocks. Mentioned blocks, from user viewpoint, represent real controllers with user interface, enabling to set their parameters. Structure of files, adjusted for Real-Time Windows Target, is shown in the following example for Sliding Mode Control:

• Structure of classical function in C code (mex function for MATLAB [1]).

Executive part of the function.
Functional parameters firstly include outputs (first line) and rest part of parameters are inputs).

```
#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 ********************************************** */
. . .
}
```

Interface of the function for MATLAB.

```
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.

```
/* 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]));
}
```

Call of executive part of the function.
Functional parameters include outputs and then inputs.

• S-function in C code (mex function for MATLAB [1]).

Example of S-function for computation of continuous robot model (SmodRob.c).

```
#define S_FUNCTION_NAME SmodRob
#define S_FUNCTION_LEVEL 2
#include <math.h>
#include <rtwintgt.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
```

Addition of the Real-Time Windows Target header, enabling to use math functions within S-functions compiled for R-T Win Target. Note: The header #include<math.h> must precede this header.

4. Real-time simulation under Windows Target

For practical use of presented control algorithms e.g. robot control under digital signal processor (DSP), we need to obtain some orientational information on time limits (minimal length of time step – of sampling period respectively), which determine time for computation of new control action.

Real-Time Windows Target, special toolbox for MATLAB, enables prototyping and testing of real-time systems; i.e. we can use a single computer, with installed MATLAB-SIMULINK environment, as a host and target. After creating a model and its simulation with SIMULINK in normal mode, Windows Target enables to generate executable code of SIMULINK schemes in a form of separately running application in a processor of host computer. The application runs in real time with SIMULINK external mode. Integration between SIMULINK external mode and Real-Time Windows Target allows to use classical SIMULINK models as a graphical user interface to active managing of process of the control during real-time run. Example of scheme for robot control under Windows Target is in Fig. 2.

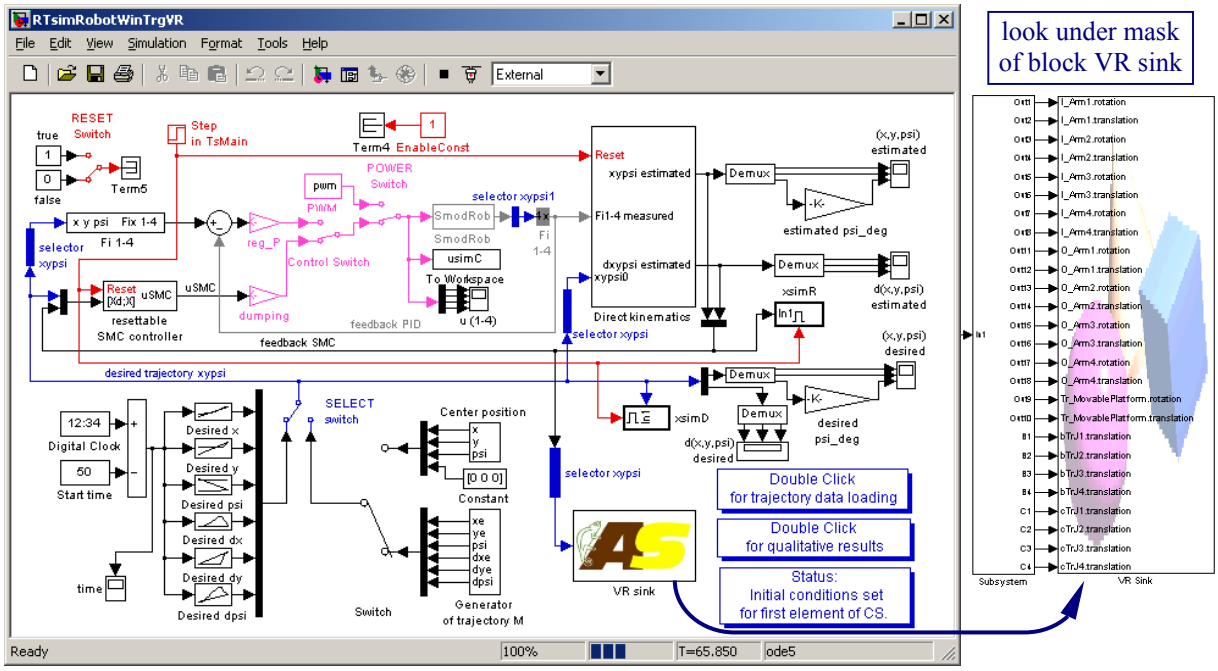

Fig. 2. SIMULINK scheme for real time simulation (with Windows Target) with visualization (Fig. 4) through Virtual Reality Toolbox (block VR Sink).

After recording and start of execution of the model in CPU we can survey needed sampling period for computing of one time step. We can also watch decrease of power of the MATLAB. It is given by outgoings on the communication between separately running application and MATLAB. (In normal simulation, the power is 100%, i.e. there is not any delay, because simulation runs directly under MATLAB). The following list shows the parameters just in real time simulation (command `>> rtwho`):

```
Real-Time Windows Target version 2.10 (C) The MathWorks, Inc. 1994-2001
MATLAB performance = 96.4%
Kernel timeslice period = 0.999 ms
TIMERS:  Number    Period    Running
             1       0.01       Yes
DRIVERS:                        Name          Address    Parameters
                          RTsimRobotWinTrgVR       0          []
```

For correct results, several parameters must be set before starting of real simulation. Following figure (Fig. 3) shows some their example. We can use choice from toolbars – – Real-Time Workshop – Options… for general setting of parameters of compilation or furthermore some choice from External mode control panel… for connection or for setting e.g. External signals & Triggering… Moreover, SIMULINK scheme in Fig. 2 includes block (VR Sink), providing connection with window of Internet Explorer, in which the given process is simultaneously visualized in simple virtual environment (in virtual reality). Example of the creation of such model will be introduced in the following chapter.

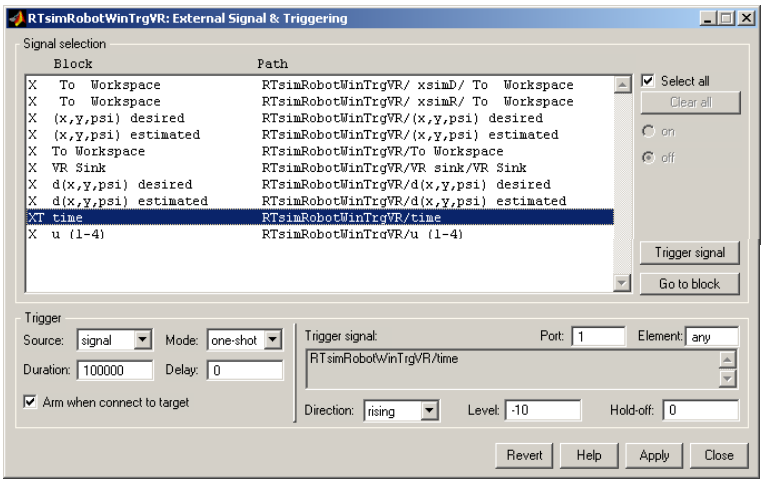

Fig. 3. Setting of External signals & Triggering.

5. Visualization with Virtual Reality Toolbox

Virtual Reality Toolbox [1] enables to construct VR model with surrounding environment in separate graphical editor (Fig. 4). Moreover, we can prepare different viewpoint directions, velocity of approach to VR object, movements etc., which can be chosen during the simulation through operating panel.

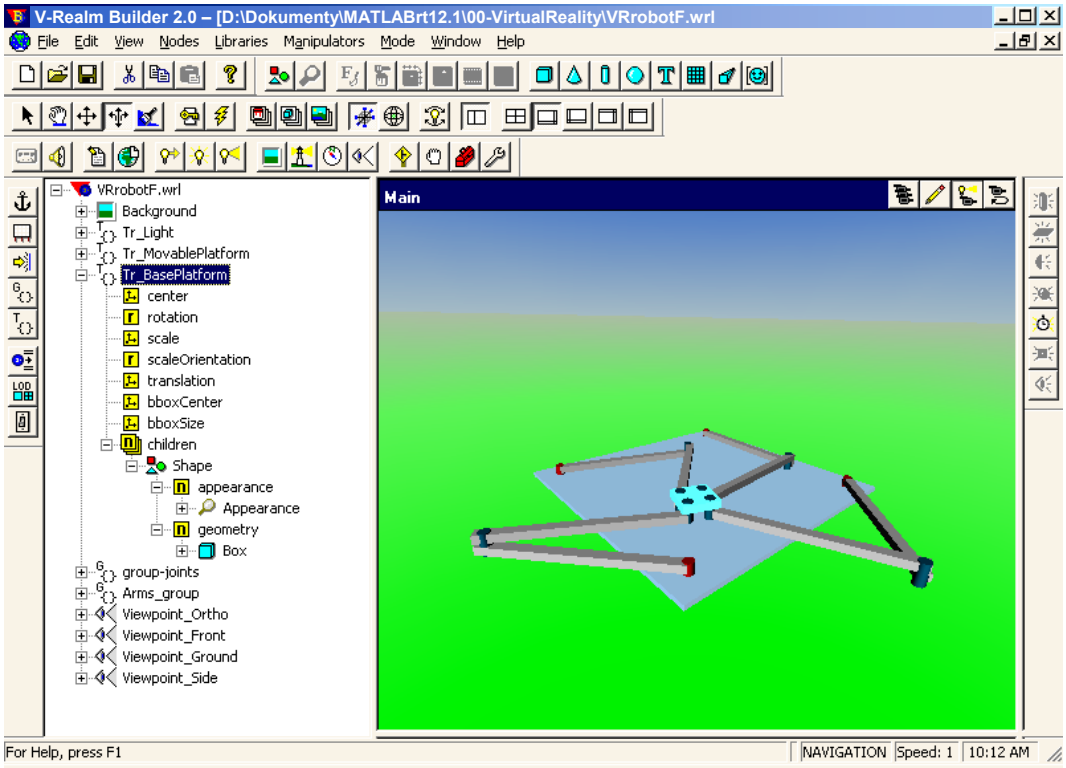

Fig. 4. V-Realm Builder 2.0 for creation of virtual reality model with its virtual environment.

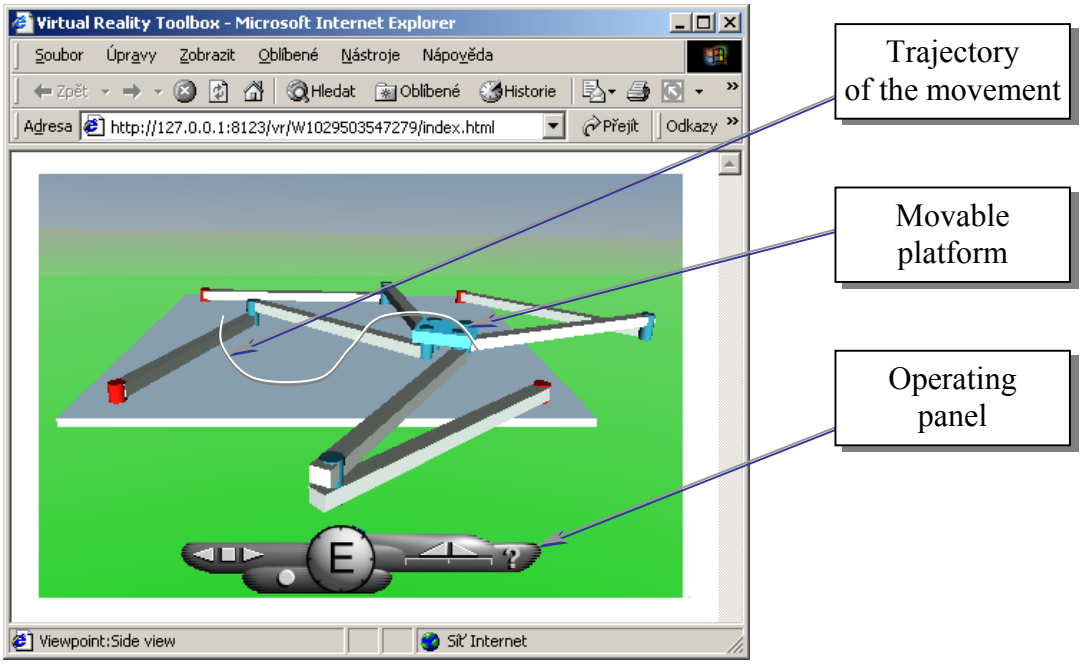

Fig. 5. Simple example of virtual reality in Internet Explorer (MS Internet Explorer) connected with SIMULINK scheme (Fig. 2) trough the Virtual Reality Toolbox.

For opening of the virtual reality window, we use the block VR Sink. It includes reference button to real window and list of all objects with their key coordinates. We mark several key coordinates, which should be changed during the process. By this fact, that the simple explorer is used, it is possible to watch given process also from remote PC.

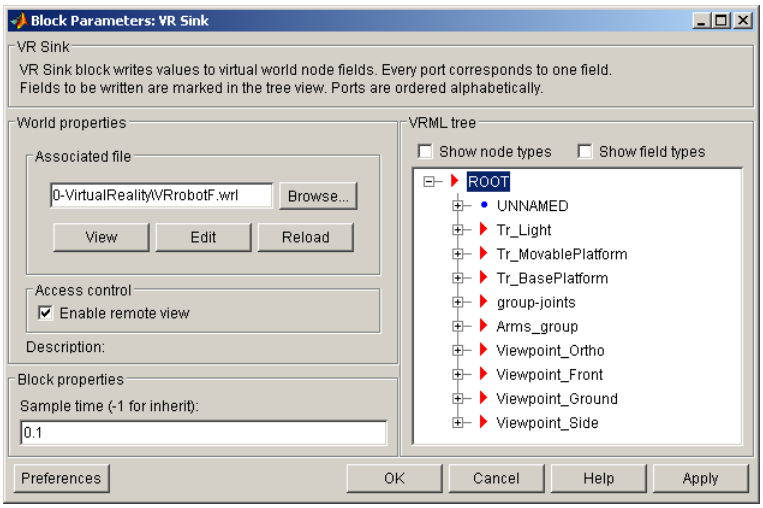

Fig. 6. User interface for opening and editing of VR model (Block Parameters: VR Sink).

At the end of this chapter let us observe, that Virtual Reality Toolbox was used only as one simple example of possibility of remote supervising of given technological process, i.e. as an inspiration for solution of the question of visualization for technological processes by means of commercially sourced visualizing programs (WinCC, InTouch, Control Panel etc.).

6. Results and Conclusion

The paper deals with C code implementation of Sliding Mode Control (SMC) and Generalized Predictive Control (GPC) as examples of high-level model-based control within SIMULINK environment. The programming in langue C is necessary for actual real-time control of the physical model of the robot by digital signal processor (DSP).

References

[1] Online Manuals (in PDF): External Interfaces; Writing S-Functions. The MathWorks, Inc. 99-01.
[2] STEJSKAL, V. – VALÁŠEK, M.: Kinematics and dynamics of machinery, Marcel Dekker, Inc. 1996.
[3] BÖHM, J. – BELDA, K. – VALÁŠEK, M.: Study of control of planar redundant parallel robot. Proceedings of the IASTED Int. conference MIC 2001, 694-699.
[4] BELDA, K. – BÖHM, J. – VALÁŠEK, M.: State-Space Generalized Predictive Control for Redundant Parallel Robots, NATO ASI Workshop, Praha 2002.
[5] VALÁŠEK, M. – STEINBAUER, P.: Nonlinear control of multibody systems, In: Euromech 99, Lisabon, 1996, pp. 437 - 444.

Acknowledgement

This research is supported by IG ČVUT (CTU IG 0204512, 2002) "Study of properties of independent (decentralized) and centralized control of redundant parallel robots" and GA ČR (102/02/0204, 2002-2004) "Design of adaptive control systems".

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