# MOTION OBSERVATION AND MODELLING IN THE VIRTUAL REALITY ENVIRONMENT

*A. Pavelka, M. Kubíček, A. Procházka*

Institute of Chemical technology, Department of Computing and Control Engineering

**Abstract**

**The paper is devoted to methods of motion observation using image acqusition toolbox followed by the mathematical modelling of the observed body including interpolation of its trajectory in the three-dimensional space. Final results are presented in the environment of virtual reality. Selected algorithms in MATLAB are included in the paper as well.**

## 1 Introduction

Analysis of videosequences and motion modelling belong to an interdisciplinary area of digital signal and image processing allowing detection, localisation, identification and prediction of moving objects components. Research of these topics include analysis of multiple marker association studied by [8], geometric algebra application analysed by [3] and methods of image processing [2, 5] including fractal analysis.

Methods of motion modelling have a wide range of applications both in engineering [7], in biomedicine [1, 6] and in further disciplines.

## 2 Data Acquisition

A synchronized two camera system for detection of precise position of a selected object in the three-dimensional space has been applied for data acquisition. Fig. 1 presents the principle of the whole system arrangement using two cameras Dragonfly connected to the PC. The Dragonfly is an OEM-style IEEE 1394 board level camera providing control and flexibility for industrial machine vision tasks. Cameras used in the system have a color CCD sensor with 1024x768 resolution and 15 frames per second. Partial Image Format (sub-sampled) allows the user to transmit a sub-sampled 640x240 image at up to 50 fps. The 6-pin 1394 standard cable provides

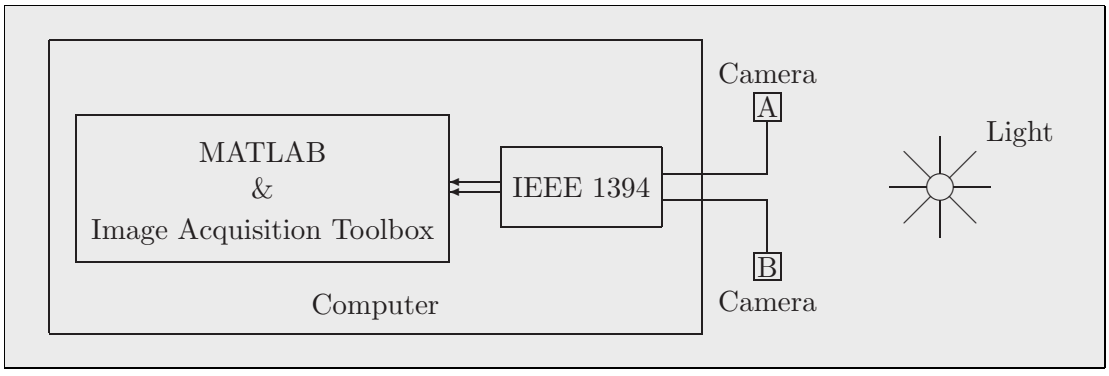


Figure 1: The two camera measurement system

the camera with both power and a connection to computer having IEEE 1394 plug-in board. The camera is supported by the original PGR Fly Capture software, but we describe its application with Image Acquisition Toolbox of the MATLAB. In the used PC Windows XP system is installed. The camera installation is described by [4].

Multiple Dragonfly's on the same IEEE-1394 bus are automatically synchronized to each other at the hardware level. The maximum deviation in the synchronisation is 125 $\mu s$ of each

other as states the Dragonfly Technical Reference Manual. This synchronisation of cameras has been verified by analysis of the record of the display of the counter counting 1 kHz pulses. It means that numbers on the display were changing thousand times per second. On the corresponding picture from each camera the same number on the display has been captured.

```
------------------------------------------------------------------------------------------
    clear all
%%% Definition of the input format %%%
    cam1=videoinput('dcam',1,'Y8_1024x768');  cam2=videoinput('dcam',2,'Y8_1024x768');
%%% The number of pictures for acquisition %%%
    deda.FramesPerTrigger=30;                 strejda.FramesPerTrigger=30;
%%% The search of the source property %%%
    src=getselectedsource(cam1);              src1=getselectedsource(cam2);
%%% Shutter speed definition %%%
    src.Shutter=5;                            src1.Shutter=5;
%%% Start images acquisition %%%
    start(cam1);                              start(cam2);
%%% Wait for the acquisition to stop %%%
    wait(cam1);                               wait(cam2);
%%% Transfer images to the MATLAB workspace %%%
  [f,t]=getdata(cam1);                        [g,t]=getdata(cam2);
------------------------------------------------------------------------------------------
```

Figure 2: Fundamental commands of the algorithm for the camera image acquisition

The system has been based upon the Image Acquisition Toolbox supporting a wide range of image acquisition operations from the professional grade frame grabbers to USB-based Webcams. The toolbox allows the connection hardware, its configuration, video preview, and transfer of the stream of images directly into the MATLAB environment for their analysis and visualization.

A short program for capture of 30 pictures by cameras and saving them to the hard disc (acquiring 15 frames per second in the monochrome mode) is presented in Fig. 2. The program for final display of differences of all saved pictures concentrated in one picture is presented in Fig. 3.

```
------------------------------------------------------------------------------------------
  k=input('Number of figures = ');
  d(:,:,1)=g(:,:,1);
%%% Compute differences between pictures %%%
  for i=1:k-1
    d(:,:,1)=imabsdiff(d(:,:,1),g(:,:,i+1));
  end
  imview(d(:,:,1),[])
------------------------------------------------------------------------------------------
```

Figure 3: Fundamental commands of the algorithm for the final image display

## 3 Three-Dimensional Modelling

The proposed system presented in Fig. 4 consists of two cameras $A$ and $B$ located in the distance $c$. The moving object $C$ represented by a light in this case form a triangle with two fixed points. In the initial stage presented in Fig. 4(a) it is possible to calibrate the system measuring distances $a_1(1), b_1(1)$ and $c$ of the triangle $ABC$. Using the cosine theorem it is then possible to evaluate initial angles $\alpha_1(1)$ and $\beta_1(1)$ by relation

$$\alpha_1(1) = \arccos \frac{b_1(1)^2 + c^2 - a_1(1)^2}{2\ b_1(1)\ c} \tag{1}$$

$$\beta_1(1) = \arccos \frac{a_1(1)^2 + c^2 - b_1(1)^2}{2\ a_1(1)\ c}. \tag{2}$$

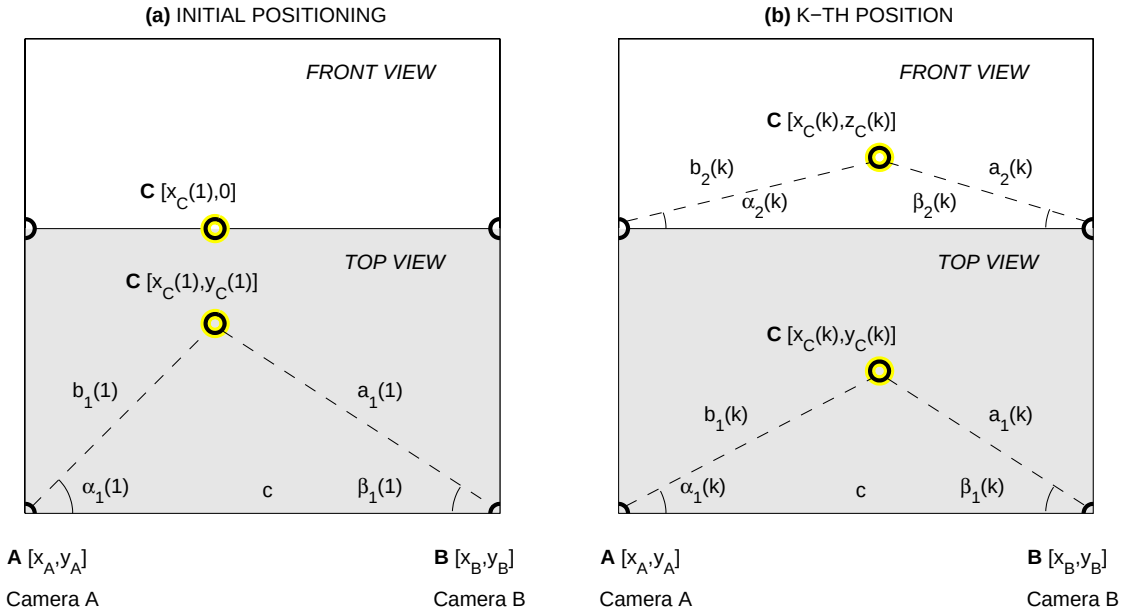


Figure 4: The samera system allowing the evaluation of **(a)** the initial localization of the light and **(b)** the three dimensional coordinates of the selected object using observations for the $k$-th pair of camera pictures

The calibration assumes evaluation of horizontal and vertical angles of both cameras. Using the calibration grid table according to Fig. 5(a) placed in the distance $d$ from each camera it is possible to find both horizontal $s_{horizontal}$ and vertical $s_{vertical}$ sizes of the figure. These parameters can then be used for evaluation of the limits of angles presented in Fig. 5(b) using rectangular red and blue triangles estimating in the case of camera $A$ values

$$\alpha_{horizontal} = 2 \arctan \frac{s_{horizontal}/2}{d} \tag{3}$$

$$\alpha_{vertical} = 2 \arctan \frac{s_{vertical}/2}{d} \tag{4}$$

and corresponding limits $\alpha_{1min}$, $\alpha_{1max}$, $\alpha_{2min}$ and $\alpha_{2max}$ using the initial point positioning and initial angles evaluated before. Similar process can be used for camera $B$.

The observation process provides precisely synchronized pictures from both cameras. The situation in the $k$-th observation step is given in Fig. 4(b). The moving object represented by a light in this case can be detected at first using a simple thresholding method applied to individual images. Using the results of calibration given in Fig. 5(b) it is possible to convert the row and column positioning of the light to horizontal $\alpha_1(k)$ and vertical angles $\alpha_2(k)$ in the case of camera $A$ and to $\beta_1(k)$ and $\beta_2(k)$ angles in the case of camera $B$.

The preprocessing of the set of camera pictures during the observations in the $k$-th observation step define precisely the triangle $ABC$ in the space with the top and front view presented in Fig. 4(b). Using the system of coordinates with the origin in the position of camera $A$ and choosing the axis $x$ in the direction of camera $B$ and $y$ axis in the plane of the initial light positioning it is possible to evaluate coordinates of point $C$ for each set of camera observations. In the top view it is possible to find coordinates of point $C$ from size $b_1(k)$ evaluated by the sine theorem in the form

$$b_1(k) = c \cdot \sin(\beta_1(k)) / \sin(\pi - \beta_1(k) - \alpha_1(k)) \tag{5}$$

$$x_C(k) = b_1(k) \cdot \cos(\alpha_1(k)) \tag{6}$$

$$y_C(k) = b_1(k) \cdot \sin(\alpha_1(k)) \tag{7}$$

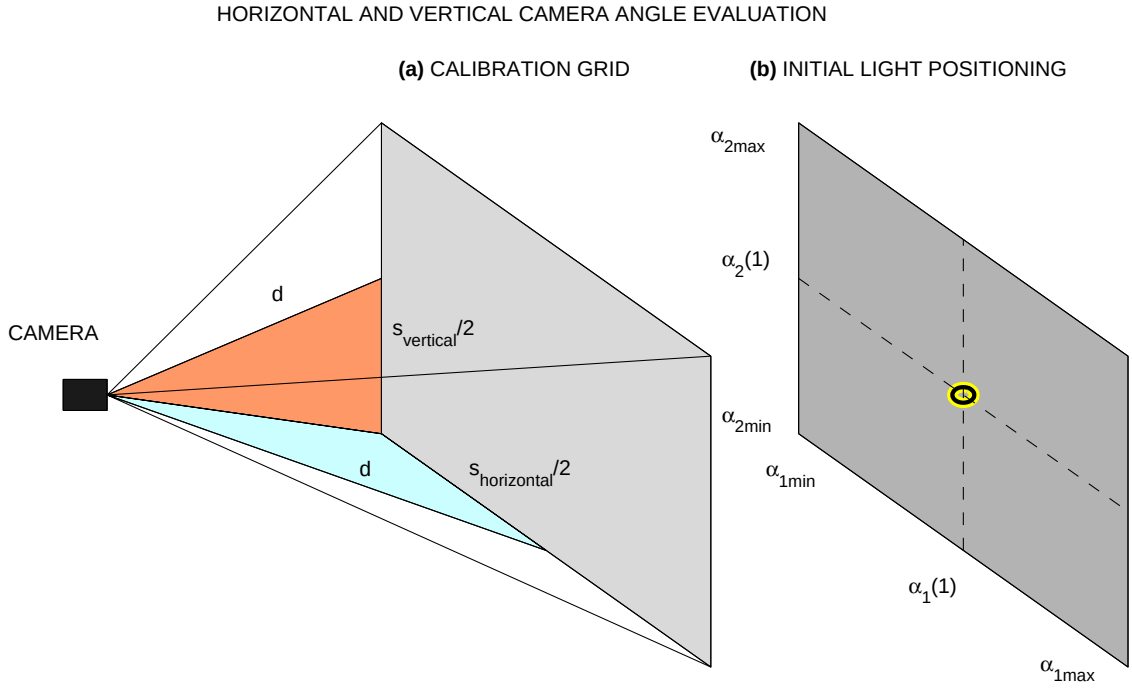


Figure 5: The principle of calibration of an individual camera **(a)** using the calibration grid table and **(b)** evaluating its horizontal and vertical angle limits allowing object localization

In the similar way it is possible to evaluate the $z$ coordinate of point $C$ using relations

$$b_2(k) = c \cdot \sin(\beta_2(k)) / \sin(\pi - \beta_2(k) - \alpha_2(k)) \quad (8)$$

$$z_C(k) = b_2(k) \cdot \sin(\alpha_2(k)) \quad (9)$$

Definition of the three dimensional positioning of the moving object is given in this way in the chosen coordinate system for each set of camera observation.

Resulting three column matrix of three coordinates of the moving object in each its line can be further processed and visualized. Results of a selected set of 30 observation is presented in Fig. 6. The following spline interpolation having similar effect as the increase of the camera frequency image acquisition can be further used to smooth results.

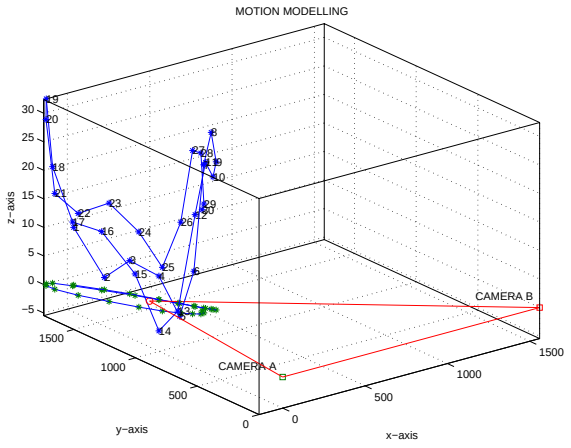


Figure 6: The three dimensional positioning of the moving object localized by its coordinates evaluated from the camera system

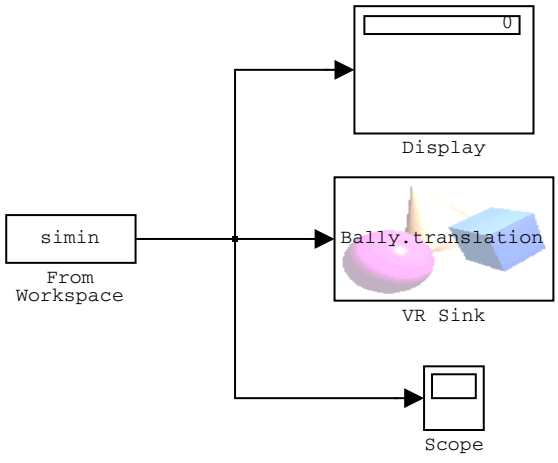


Figure 7: The Simulink model enabling the study of the moving object in the environment of the virtual reality

# 4 Virtual Reality Use for Motion Visualization

Virtual reality (VR) environment can be used very efficiently for motion modelling and for the study of the moving object properties. This kind of modelling enables definition of different objects that can be included in the final model to make the scenery as realistic as possible. Moreover it enables the incorporation of results of mathematical or physical modelling to make movement corresponding to real situations. To build such a model in the case under study it is possible to use results of data acquisition and processing in the MATLAB environment presented above. The final image can be transformed into a VR model whose objects can be controlled from the Simulink environment presented in Fig. 7. The data set evaluated in MATLAB can be used to control the specific movement of those objects.

The proposed VR model and its control system in Simulink is created as the whole in the MATLAB environment. For the user of the program it is necessary to capture the image sequence only. The rest of the process including the image identification, calculation of position of the light, spline interpolation of its movement, Simulink model creation and the final VR model creation is fully automatic. The creation of the VRML file using results of MATLAB modelling with the Virtual Reality Toolbox is enabled by the `vrml(gcf,name_file)` command.

```
%% VRML Creation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(5)
% Trajectory of moving object
    plot3(siminin(:,1),siminin(:,2),siminin(:,3)); hold on;
% Zero Camera Plane
    plot3([0 c xc0 0],[0 0 yc0 0],[0 0 zc0 0],'r'); hold off
% Removing useless graphics
    axis tight; box on
    set(gca,'XTick',[]); set(gca,'YTick',[]); set(gca,'ZTick',[]);
% Correction of camera position
    set(gca,'CameraPosition',get(gca,'CameraPosition')/1.8);
% Generating of unique file name
    name_file = ['ball_',num2str(now),'.wrl'];
    name_file = [name_file(1:11) name_file(13:end)];
% Create VRML file
    vrml(gcf,name_file);
```

Figure 8: Algorithm for the simple VRML creation

Fig. 8 describes how easy it is to create the creation of the VR model. The most complicated problem in the whole process is in the calculation and plot of a simple and transparent graph without to much detail objects as shown in Fig. 9. Fig. 10 has been created from the classical MATLAB Fig. 9 using the `vrml(gcf,'my_vrml_file.wrl')` command only having in mind that the MATLAB's Virtual Reality Toolbox is necessary for its execution. The comparison of Figs 9 and 10 show no difference between MATLAB figure and VRML model not taking into account that the VRML viewpoint is sightly changed only for the better comparison according to the original viewpoint.

Commands enabling operations with files (`fopen`, `fread`, `fprintf` and `fclose`) - in our case with simple text files - have been used for special modifications like the addition of the light ball, camera models creation or definition of the scenery highlight in the VR environment. Fig. 11 represents creation of the yellow sphere (ball) imaging the moving light.

The first part of the code shows calculation of parameters (scale, translation and others) from the given figure. The second part, definition of the `bally` variable that contain specification of the sphere in the VRML language with added values of parameters corresponding to our problem. The same principle has been used in modelling of cameras A and B. Pre-modelled cameras have been added from their source text files (`camera_A.txt` and `camera_B.txt`) and specifically modified to create more realistic VRML scene. For example on the forth line of

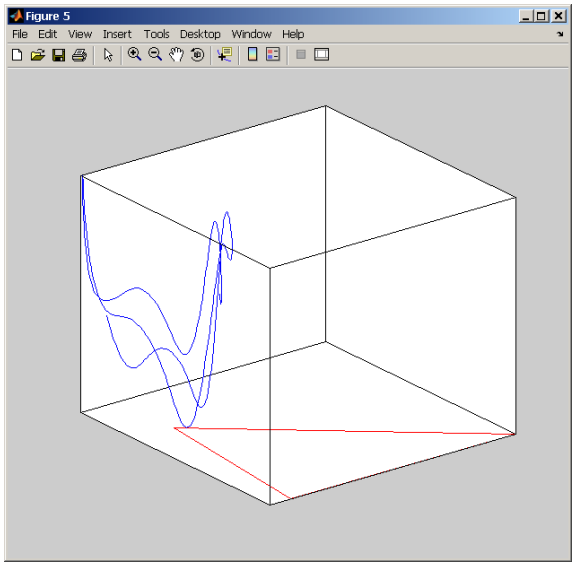


Figure 9: Figure for VRML model creation

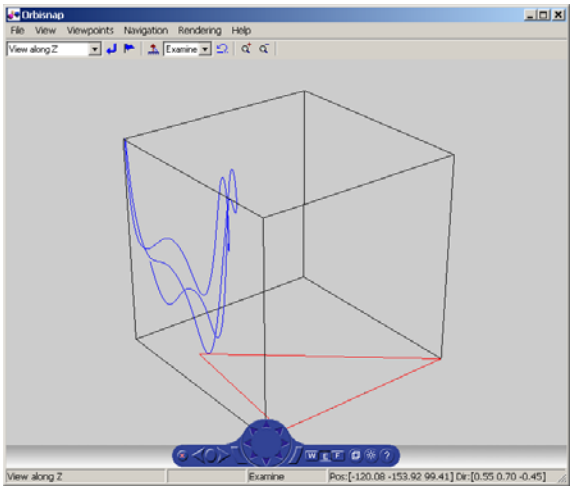


Figure 10: Created VRML model

the code shown in Fig. 12 the word FALSE has been changed to the word TRUE. This change enables to see the whole VRML scene in colors and with textures. Models of both cameras have been prepared separately and they are saved in plain text files in the VRML language. The program reads those files and place their content to the specific place.

The same principle has been used during the creation of the corresponding Simulink model (Fig. 7) from the template (`bally_mdl01.txt` and `bally_mdl02.txt`) with its code in Fig. 13. This template has been programmed before and it enables the movement of the light (the yellow ball) on the trajectory. The object of the light in the VR model called `Bally` has the property `translation` that determines the position of the object and this property is changed in the VR model by Simulink in the real time.

```
-------------------------------------------------------------------------------------
% Yellow Ball in VRML file
obj = get(gca);
scaleXYZ = obj.PlotBoxAspectRatio./obj.DataAspectRatio;
translationXYZ = [-sum(obj.XLim)/2 -sum(obj.YLim)/2 -sum(obj.ZLim)/2];
position2 = obj.CameraPosition;
fieldOfView = obj.CameraViewAngle*pi/180;
scaleRatio = 1;

bally = ['DEF Bally Transform {',char(10)...
char(9),'translation    ',num2str(siminin(1,:)),char(10)...
char(9),'scale  ',num2str(1./scaleXYZ*scaleRatio),char(10)...
char(9),'children Shape {',char(10)...
char(9),char(9),'appearance Appearance {',char(10)...
char(9),char(9),char(9),'material Material {',char(10)...
char(9),char(9),char(9),char(9),'ambientIntensity   0.6',char(10)...
char(9),char(9),char(9),char(9),'diffuseColor   1 1 0',char(10)...
char(9),char(9),char(9),char(9),'emissiveColor 0.6 0.4 0',char(10)...
char(9),char(9),char(9),char(9),'shininess 0.2',char(10)...
char(9),char(9),char(9),char(9),'specularColor 0 0.4 0',char(10)...
char(9),char(9),char(9),char(9),'transparency 0.2}}',char(10)...
char(9),char(9),'geometry   Sphere {}',char(10)...
char(9),'}}',char(10)];
-------------------------------------------------------------------------------------
```

Figure 11: Algorithm for the creation of the yellow ball in VRML

```
% Reading VRML file and setting Highlight 'TRUE ' ([84 82 85 69 32])
end_file = [32 32 32 93 10 32 32 125 10 32 93 10 125 10]';
fid = fopen(name_file,'r'); wrmlbi=fread(fid,'char'); fclose(fid);
wrmlbi = wrmlbi(1:length(wrmlbi)-length(end_file));
wrmlbi(77:81) = [84 82 85 69 32]';

% Reading and setting of Cameras
fid = fopen('camera_A.txt','r'); camera_A=fread(fid,'char'); fclose(fid);
fid = fopen('camera_B.txt','r'); camera_B=fread(fid,'char'); fclose(fid);

camera_A_mod = [char(9),'rotation   ',num2str([0 0 1 alpha10-pi/2]),char(10)...
                char(9),'scale  ',num2str(1./scaleXYZ*scaleRatio),'}',char(10)];

camera_B_mod = [char(9),'rotation   ',num2str([0 0 1 pi/2-beta10]),char(10)...
                char(9),'scale  ',num2str(1./scaleXYZ*scaleRatio),char(10)...
                char(9),'translation    ',num2str([c 0 0]),'}',char(10)];

% Save of modified VRML file with Highlight and Yellow Ball
fid = fopen(['bally',name_file(6:end)],'w');
fprintf(fid,'%s',[char(wrmlbi); bally'; char(camera_A); camera_A_mod';...
    char(camera_B); camera_B_mod'; char(end_file)]);
fclose(fid);
```

Figure 12: Algorithm for the placing of camera models in VRML scene

The main goal of the MATLAB's code in Fig. 13 is to put together opening and terminal part of Simulink code and to tie them with the corresponding unique name of `WorldFileName` (name of `*.wrl` file) for connection between Simulink and VRML. The final part of the code prepare the data set for the object movement. This data set is interpolated by the spline curve representing trajectory of the moving object and its values are saved in the `*.mat` file with the unique file name.

The last presented part of the code shows in Fig. 14 the method of the activation of the arbitrary object in the VRML world. In the first half of the code each row of the matrix `simin.signals.values` with coordinates of interpolated trajectory is decoupled according to the value of parameter `k`. This parameter has the speed affect to the movement of the sphere. The middle part of the code defining the `animation` variable represents the main part that is in this step added to the new VRML file with the moving object. The code is closed as usually by saving the results into the plain text file.

```
%% MDL Creation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reading MDL from template
fid = fopen('bally_mdl01.txt','r'); bally_mdl01=fread(fid,'char'); fclose(fid);
fid = fopen('bally_mdl02.txt','r'); bally_mdl02=fread(fid,'char'); fclose(fid);

% Save MDL with specific *.wrl file
fid = fopen(['bally',name_file(6:end-3),'mdl'],'w');
fprintf(fid,'%s',[char(bally_mdl01); ['bally',name_file(6:end)]'; char(bally_mdl02)]);
fclose(fid);

% Structure variable simin for MDL
siminin = [xxc' yyc' zzc'];
simin.signals.values = [ones(50,1)*siminin(1,:); siminin; ones(50,1)*siminin(end,:)];
simin.time=[];
save(['bally',name_file(6:end-3),'mat'],'simin')
```

Figure 13: Algorithm for the dynamical creation of Simulink model

```
%% VRML Creation with Animation %%%%%%%%%%%%%%%%%%%%%%
% Calculation of animation
k = 10; % setting of animation speed (higher k = slower animation)
s = simin.signals.values;
L = size(s);

KEY = ((1:k*L(1))-1)/(k*L(1)-1);
KEYVALUE = zeros(k*L(1),3);
for i = 1:3
    KEYVALUEm = ones(k,1)*s(:,i)';
    KEYVALUE(:,i) = KEYVALUEm(:);
end

KEYVALUEstr = [];
for i = 1:k*L(1)
    KEYVALUEstr = [KEYVALUEstr,char(9),char(9),num2str(KEYVALUE(i,:)),',',char(10)];
end
KEYVALUEstr = KEYVALUEstr(1:end-2);

% Definition and calculation of the ball motion
animation = ['DEF Replay_control TimeSensor {',char(10)...
  char(9),'cycleInterval  10',char(10)...
  char(9),'loop   TRUE',char(10)...
  '}',char(10),char(10)...
  'DEF Bally_translation_recorded PositionInterpolator {',char(10)...
  char(9),'key',char(9),'[',char(10)...
  num2str(KEY),']',char(10)...
  char(9),'keyValue',char(9),'[',char(10)...
  KEYVALUEstr,']',char(10)...
  '}',char(10)...
  'ROUTE Bally_translation_recorded.value_changed TO Bally.translation',char(10)...
  'ROUTE Replay_control.fraction_changed TO Bally_translation_recorded.set_fraction'];

% Reading VRML file
fid = fopen(['bally',name_file(6:end)],'r'); wrmlbi=fread(fid,'char'); fclose(fid);

% Save of modified VRML file with animation of the Yellow Ball
fid = fopen(['ballya',name_file(6:end)],'w');
fprintf(fid,'%s',[char(wrmlbi(1:k(2)+23)); num2str(m)';
    char(wrmlbi(k(2)+23+l(1):end)); animation']);
fclose(fid);
```

Figure 14: Algorithm for the VRML animation of the moving sphere

System presented in the virtual reality environment can be studied and observed from the arbitrary location of the viewer in the space using any zoom level. Selected results are presented in Figs 15 up to 19 showing different views towards the moving object.

Real computer system enables even more realistic view and the study of the motion enabling control of the motion speed as well. System described above presents a very simple example of possibilities of three dimensional motion modelling following one moving object only.

Similar approach can be used in the case of detection of several moving objects allowing the study of their interaction as well.

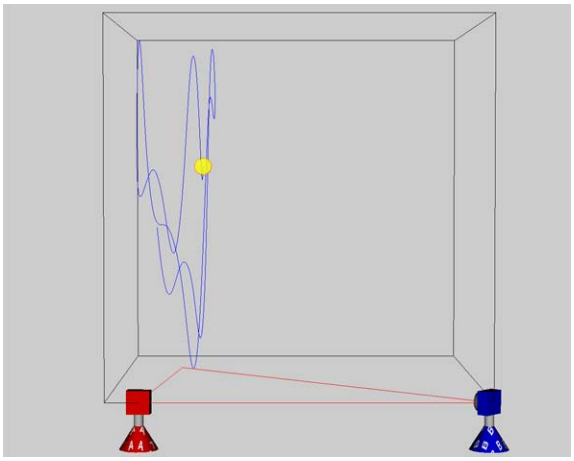

Figure 15: The back view of the system

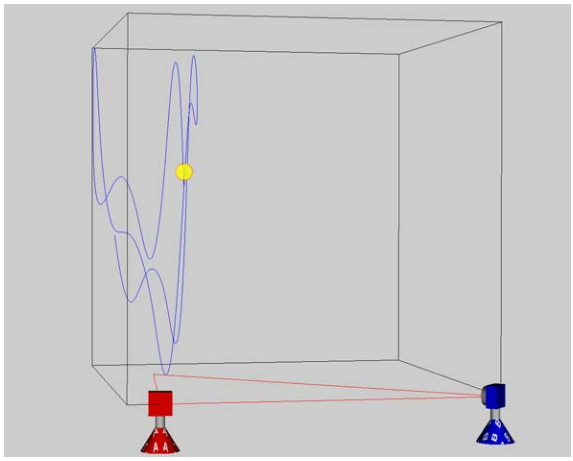

Figure 16: The view towards the moving object from the position of camera A

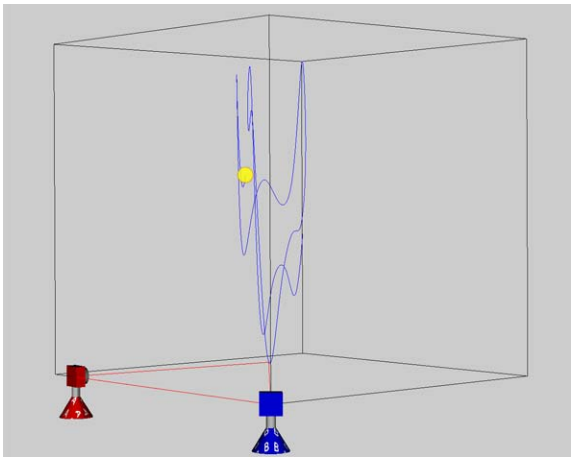

Figure 17: The view towards the moving object from the position of camera B

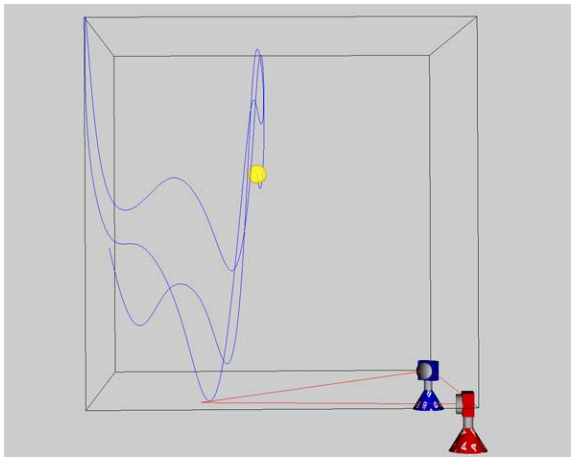

Figure 18: The side view of the system

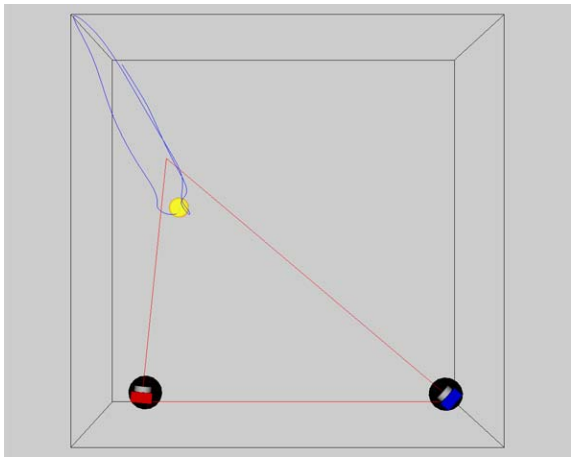

Figure 19: The top view of the system

## 5 Conclusion

The paper presents both technical principles of the moving body detection using the image acquisition toolbox and a synchronized two video cameras system. Specific mathematical methods of image components localization for processing of each couple of observed images are then used.

Basic principles of the virtual reality visualization presented in the paper enable a very convenient way of the study of a moving object both in the simple example presented in the paper and in more complex applications. The paper presents how such a model based on the MATLAB computational system can be created. The most important algorithms are included in the paper as well.

## Acknowledgments

The work has been supported by the research grant of the Faculty of Chemical Engineering of the Institute of Chemical Technology, Prague No. MSM 6046137306.

## References

[1] R. Boulic, P. Fua, L. Herda, M. Silaghi, J.S. Monzani, L. Nedel, and D. Thalmann. An Anatomic Human Body for Motion Capture. In *Technologies for the Information Society: Developments and Opportunities*. EMMSEC98, 1998.

[2] R. C. Gonzales, R. E. Woods, and S. L. Eddins. *Digital Image Processing Using MATLAB*. Prentice Hall, 2004.

[3] J. Lasenby and A. Stevenson. Using Geometric Algebra for Optical Motion Capture. In E.Bayro-Corrochano and G. Sobcyzk, editors, *Applied Clifford Algebras in Computer Science and Engineering*. Birkhauser, Boston, U.S.A., 2000.

[4] M. Kubíček. Using Dragonfly IEEE-1394 Digital Camera and Image Acquisition Toolbox. In *Sborník konference MATLAB 2004*, pages 280–282. VŠCHT Praha, 2004.

[5] M. Nixon and A. Aguado. *Feature Extraction & Image Processing*. NewNes Elsevier, 2004.

[6] M. Ringer, T. Drummond, and J. Lasenby. Using Occlusions to Aid Position Estimation for Visual Motion Capture. In *Proc Computer Vision and Pattern Recoginition (CVPR)*. IEEE USA, 2001.

[7] M. Ringer and J. Lasenby. Modelling and Tracking of Articulated Motion from Multiple Camera Views. In *Proc. British Machine Vision Conf (BMVC)*, pages 172–181, 2000.

[8] M. Ringer and J. Lasenby. Multiple Hypothesis Tracking for Automatic Optical Motion Capture. *Lecture Notes in Computer Science*, 2350, 2002.

---

Ing. Aleš Pavelka, Assoc. Prof. Miroslav Kubíček, Prof. Aleš Procházka
Institute of Chemical Technology, Prague
Department of Computing and Control Engineering
Technická 1905, 166 28 Prague 6 Phone: 00420-220 444 027, 00420-220 444 198
E-mail: Ales.Pavelka@volny.cz, Miroslav.Kubicek@vscht.cz, A.Prochazka@ieee.org
Web: http://dsp.vscht.cz