

CONTROLLING A VIRTUAL
MARIONETTE USING A WEB
CAMERA

by

Alex Sirota

Dov Sheinker

Oren Yossef

Project in intelligent systems, 236754
Computer Science Department
Technion – Israel Institute of
Technology

Under supervision of:
Professor Alfred Bruckstein, Adi Bar-Lev

September 2003

TABLE OF CONTENTS

1.	INTRODUCTION.....	1
2.	SOLUTION AND ALGORITHMS	3
2.1.	GENERAL.....	3
2.2.	IMAGE SEGMENTATION AND LOCATING THE OBJECT.....	3
2.3.	ORIENTATION AND TRANSLATION RECONSTRUCTION	4
3.	SOFTWARE ARCHITECTURE	7
3.1.	GENERAL.....	7
3.2.	SOFTWARE LIBRARIES USED	7
3.2.1.	<i>Microsoft's VisionSDK</i>	7
3.2.2.	<i>Carnegie Mellon University Vision Library (CMVision)</i>	7
3.3.	SOFTWARE MODULES	7
3.3.1.	<i>Class CWatcher</i>	7
3.3.2.	<i>Class CRecon</i>	8
3.3.3.	<i>MarioLibDemo Program</i>	9
3.4.	FLOW.....	10
4.	EXPERIMENTS	11
4.1.	GENERAL.....	11
4.2.	VISION.....	11
4.3.	RECONSTRUCTION.....	12
5.	RESULTS AND CONCLUSIONS.....	15
5.1.	RESULTS.....	15
5.2.	PROBLEMS AND ENHANCEMENTS	17
6.	REFERENCES.....	18
8.	APPENDIX A: THE HSV COLOR MODEL	19
9.	APPENDIX B: MATLAB IMPLEMENTATION OF THE RECONSTRUCTION	20
10.	APPENDIX C: OLD ALGORITHM FOR ORIENTATION AND TRANSLATION RECONSTRUCTION	24

1. INTRODUCTION

The problem we were trying to solve is to manipulate a virtual 3D object like a marionette using a real world 3D object and a camera.

The real object image, as received from the camera is analyzed, the object is located in the image and its orientation and position are determined. The orientation and position of the real object determine the orientation and position of the virtual object – marionette.

The problem as stated has two main parts – first, given a color image, effectively locate the object in it. Then, given the 2D projection of the object on the image plane, determine it's 3D orientation and position.

2. SOLUTION AND ALGORITHMS

2.1. General

One of the requirements of the project was that the camera be a simple web camera. We used Veo's Velocity Connect web camera, which is a pretty standard web camera.

The real object we used is a planar cross, having colored balls at its edges. The colors of the balls are (in a clockwise order): blue, red, green, red. The dimensions of the cross are approximately 28cm x 28cm.

The cross itself is white, and the both the background and the hand holding the cross should be white. A white wall and a white glove work well.

2.2. Image segmentation and locating the object

We locate the object in the image using the four colored balls at its edges. To effectively segment the image using the color information, we convert the input RGB image to HSV format (see appendix). This format is much more suitable for color analysis, because we have a separate channel (hue) which denotes the color, whereas in RGB the color information is a 3D space. HSV also separates the color information (hue) from its purity (saturation) and brightness, allowing the segmentation process to be more invariant to lighting conditions and thus more robust.

Once we have an HSV image, we perform a connected components analysis on it. Each ball color (red, green, blue) is given its own threshold range for the Hue channel corresponding to the hue range of the color. We want colorful pixels, so the threshold range for the Saturation channel is the upper

part of the range (i.e. 80-255). Finally, we consider pixels of all Brightness values.

We go over all the image pixels and classify them as one of the balls (if they fall inside the threshold) or as background. This gives us a “map” – an image having 4 possible values. On that image we run a connected components algorithm, which for each ball gives us a list of all found connected components sorted by their size in a descending order and having their centroid info attached.

The green/blue balls are located at the centroid of the largest green/blue component. The two red balls are located at the centroids of the two largest red components.

We now have four coordinates, but we still have to disambiguate the two red balls. We construct three vectors: Green-Blue, Green-Red1, Green-Red2 and perform a cross product between the first vector and the other two. The signs of the cross products allow us to disambiguate the first red ball from the second.

Thus, we have obtained the four ball coordinates in the image.

2.3. Orientation and translation reconstruction

The task here is to recover the rotation 3×3 matrix and the translation 3×1 vector given the 4 correspondence points between the image plane and the object plane (note that the object is necessarily planar, so the coordinates in the object space are also 2D). See [Bruckstein 99] for more details.

We used a weak perspective projection model, which assumes that the object is viewed from sufficiently large distance. In this case, the perspective equations:

$$x'_i = \frac{X'_i f}{Z'_i} \quad y'_i = \frac{Y'_i f}{Z'_i}$$

become more simple:

$$x'_i = \alpha X'_i \quad y'_i = \alpha Y'_i$$

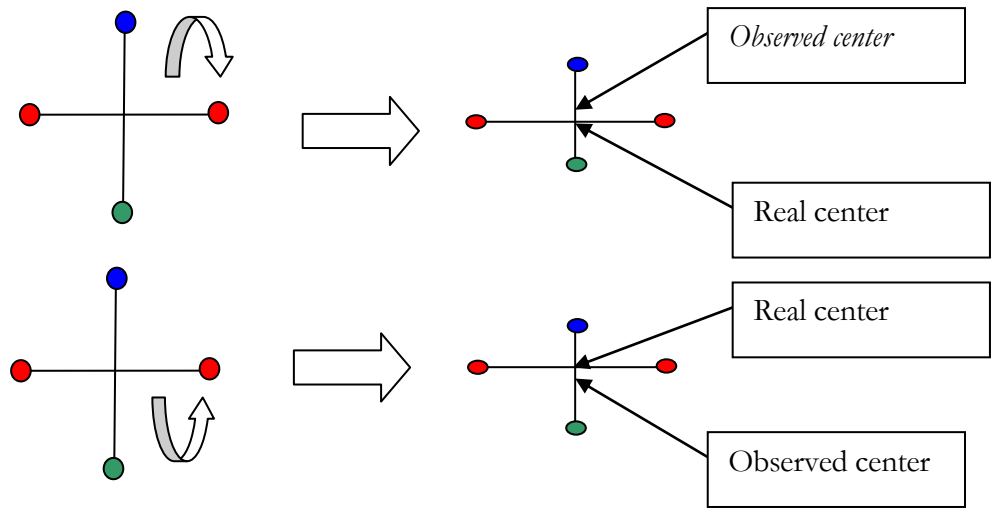
for some positive constant α ($\alpha = f/t_3$, where t_3 is the displacement of the object).

These simplified equations allow a closed form algebraic solution presented in [Bruckstein 99], section 3.2.

One big advantage of this method is that the recovery of rotation when the data is noisy is the same as in noise-free case. This is a consequence of having enough information to determine the unknowns. The recovered rotation matrix will always be orthogonal.

The equations give two solutions for the equation, as is generally the case with orthographic and weak perspective projection (duality).

We solve the duality problem using the following procedure. The object we are observing is a planar cross. The intersection of its two axes in image space gives us the center point of the cross. Under perspective projection, the observed center of the axis is not the real center. Having the knowledge of both real center and the observed one can help us to disambiguate cross pose.



As can be seen in the diagram, in the first case, the observed center is closer to the blue ball than the real center, which means that the blue ball is closer to the viewer. In the second case, the opposite situation is shown.

We calculate the distance between the real and the observed centers for both axes. If this difference exceeds the threshold, we use the axis having the bigger difference for disambiguation.

3. SOFTWARE ARCHITECTURE

3.1. General

This paragraph deals with the system implemented, its specific modules and data flows between them.

3.2. Software Libraries Used

Several software libraries were used:

3.2.1. Microsoft's VisionSDK

This software package is used to interact with the web camera or AVI file for acquiring images.

3.2.2. Carnegie Mellon University Vision Library (CMVision)

Carnegie Mellon University

Author: James R. Bruce

This software package was heavily revised by us to support the HSV color model. It is used for segmentation and connected component detection in the process of locating object in the image.

3.3. Software Modules

Main modules of our program

3.3.1. Class CWatcher

This class is responsible for analyzing the image from the web camera/AVI file to produce the four image coordinates, which are input for the reconstruction process. The theoretical grounds for this class' operation are laid in section 2.2.

The class procedure is as follows:

- 3.3.1.1. A given input image in the RGB color model is converted to the HSV color model. There are three modes for the conversion:
 - 3.3.1.1.1. - Real time conversion (CPU consuming)
 - 3.3.1.1.2. - Full lookup conversion (memory consuming)
 - 3.3.1.1.3. - Quantized lookup conversion (CPU and memory conserving, some of the precision is lost when quantizing 256 levels into 64)
- 3.3.1.2. CMVision library (see section 3.2.2) is used with proper threshold values to remove noise and isolate desired hues (balls' hues).
- 3.3.1.3. CMVision library is used to find connected components on image after threshold (segmentation).
- 3.3.1.4. Four coordinates are extracted from the connected components (one for each hue). These coordinates are the coordinates used as input for the reconstruction process

3.3.2. Class CRecon

This class is responsible for reconstructing a 3D transformation from four image coordinates. This class uses MTL (see section 1.1.1) for matrix algebraic.

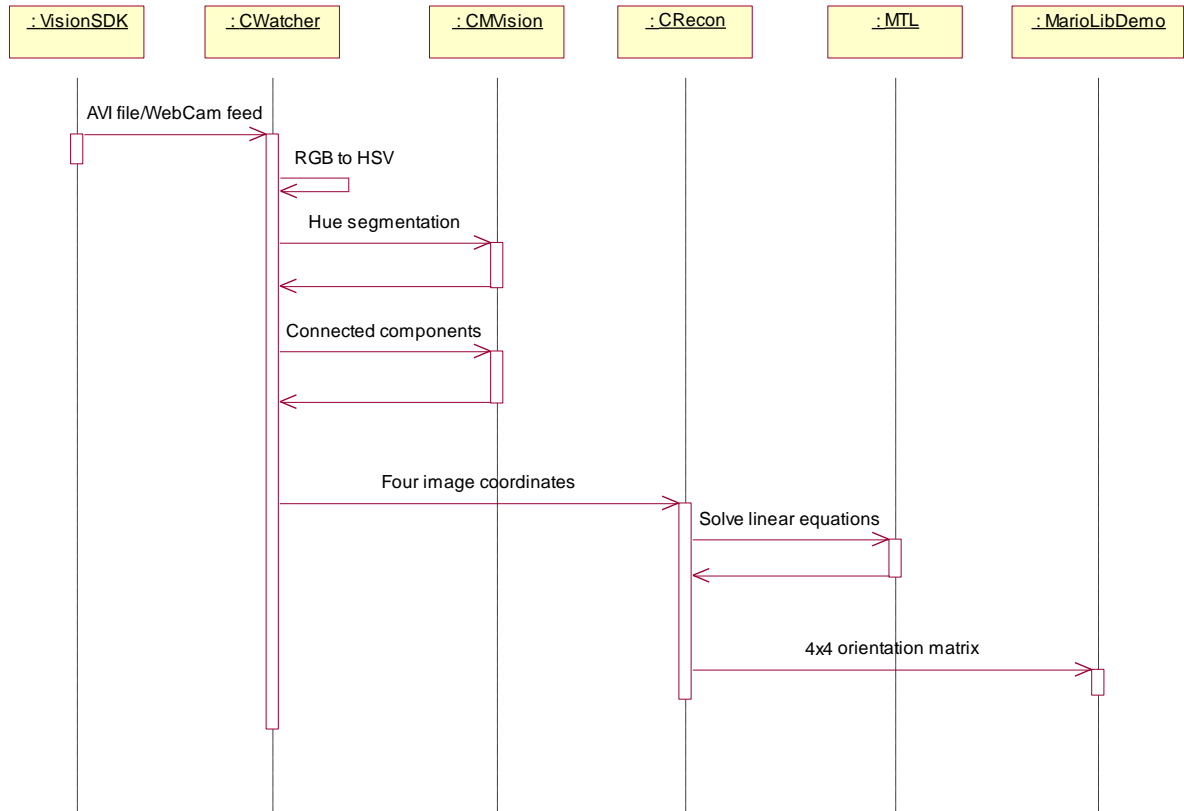
Class' initialization is done according to a proper focal length value (web camera's parameter) and four coordinates which describe the planar object. After each image acquisition and analysis, the four image coordinates, which are extracted using CWatcher (see section 3.3.1) are sent to this class for

reconstruction. As a result a 4x4 orientation matrix is returned as output of this process, this matrix is the translation and rotation of the 3D object according to the given image coordinates in relation to the four coordinates received in the initialization which describe the planar object. The theoretical grounds for this class' operation are laid in section 2.3.

3.3.3. MarioLibDemo Program

A client program for MarioLib.DLL. This program uses MarioLib.DLL to retrieve the 4x4 orientation matrix and displays a 3D planar cross with the 4x4 orientation matrix applied. For display the program uses Microsoft's DirectX 9.0.

3.4. Flow



4. EXPERIMENTS

4.1. General

This project had undergone many changes till the last version has been finalized. We have tried several algorithms in order to strive for the solution, and read many articles trying to find the right way to handle the problem.

Our experiments can be divided into two major topics. At first, we tried to get good results for the vision part of the project. On the second part, we concentrated on trying to find the right way to reconstruct the 3D orientation from the 2D image.

4.2. Vision

At the beginning of the project we thought of using a simple RGB analysis. The idea was to use balls colored in red, green, or blue, each placed at a different edge of a white cross, with a white background. It turned out that the RGB format is not so simple in terms of color space and is very sensitive to lighting conditions. Therefore we got different results every time we tested it.

Looking for a more stable format, we tried to use the YUV color format. The image vision part of the code was implemented using CMVision library. We got much better results but still it was not sufficient. It seemed that since the web camera was of very low quality and lighting conditions weren't stable enough, the analysis should have been very clever to identify the colors (at times, even we could not identify the colors when looking at the camera's image).

In order to be totally insensitive to light changes, we tried to use LEDs instead of the balls. We put 4 LEDs on the edges of the cross, each with a unique color. We created complete darkness in the room and then activated the web camera. The image analysis was still using YUV. The idea was that we can also create complete darkness in any other room. This way, the colors that the camera should identify will be equal in every room and it will be simpler to analyze them.

There were two problems with this idea. The first was that the LEDs' light was different in its intensity and sensitive to electricity conditions (battery power, connections' resistance, etc.). The second problem was that there was a blur in the image taken by the web camera when the cross was in motion. We could overcome the first problem, but since we did not find any way to overcome second problem, the idea was dropped.

We turned back to the balls and tried to make it more stable. We replaced the cross and the balls to more vivid and less shining colors. Then we used the HSV image format instead of YUV because it is less sensitive to light changes. When using the HSV we determined the threshold according to colors of the balls rather than the intensity. In order to use the HSV format we rewrote the CMVision library to work with HSV space. We used Photoshop to get the hue ranges of each color and made some tests using Matlab to test the new vision algorithm. When we saw that the results were pretty good we implemented it in C++ and add some more minor color tunings to get the best results.

4.3. Reconstruction

At the beginning of the project we thought that it would not be too complex to solve the problem of reconstruct a 3D object from a 2D image using the fact that we know the object dimensions. We tried to solve the problem using naïve methods with our information about the cross object. We know that we

need to find 4 3D points and the data we have is: 4 2D points from the camera, the fact that all 3D points are on the same plane, we know the real distance between any 2 points, and we know that the points create orthogonal lines. When we used all this information we got some complex non-linear equations that we could not solve even with the help of some professional programs (MATLAB)

Then we tried to define the Euler angles as the unknowns. We had problems of non-uniqueness of the decomposition to the three angles, singularities etc.. Reading a lot of material on the subject we came to an understanding that Euler angles are good for specifying small incremental rotation, but are very problematic specifying orientation.

We also considered a solution using artificial intelligence, which would try to guess the movement of the cross according to its former location. By calculating a tree of all the possibilities, the algorithm would pick the closest state, which is similar to in its 2D dimensions to the image we've got from the web camera. After some short attempts we did not think it was practical.

We also thought about a solution based on equations using Quaternion representations. The advantage of the quaternion representation is that it directly displays the intrinsic geometric properties of the rotation--its axis and angle--and moreover has all the algebraic information we need to compute anything we need to compute about the rotation.

We read many articles trying to see if anyone had already dealt with a problem like ours and to figure out what was wrong with our former solutions. Eventually we found [Horn 2] article, which solved a very similar problem. We took the equations from this article and in order to test their correctness for the matrix reconstruction of the rotation, we created a project in matlab that tests it mathematically and visually.

A description of the matlab project is described in [APPENDIX B: MATLAB implementation of the reconstruction].

When we saw that the code solves the problem, we implemented it in C++. The description is in [APPENDIX C: Old algorithm for Orientation and translation reconstruction].

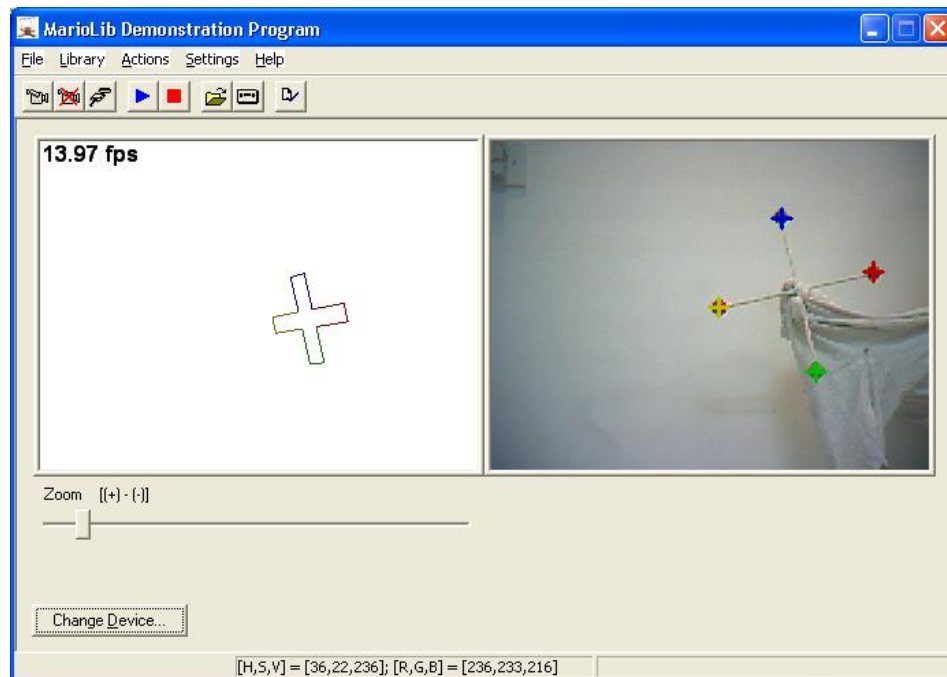
After implementing the code in C++ and testing it with our application we noticed that the reconstruction isn't working as expected in some cases. After investigation we discovered that the rotation matrix that is reconstructed isn't orthogonal, which gives some unusual effects when applied to the object. We began looking into algorithms for making the obtained rotation matrix orthogonal. There are many methods of doing this, many of which involve iterative process of taking the initial guess and refining it until a orthogonal matrix is obtained. While choosing a appropriate algorithm, we tried the simplified weak perspective model described in [Bruckstein 1] and found out that it solves the problem nicely in our case. We tested the proposed algorithms in MATLAB as earlier and then implemented it in C++.

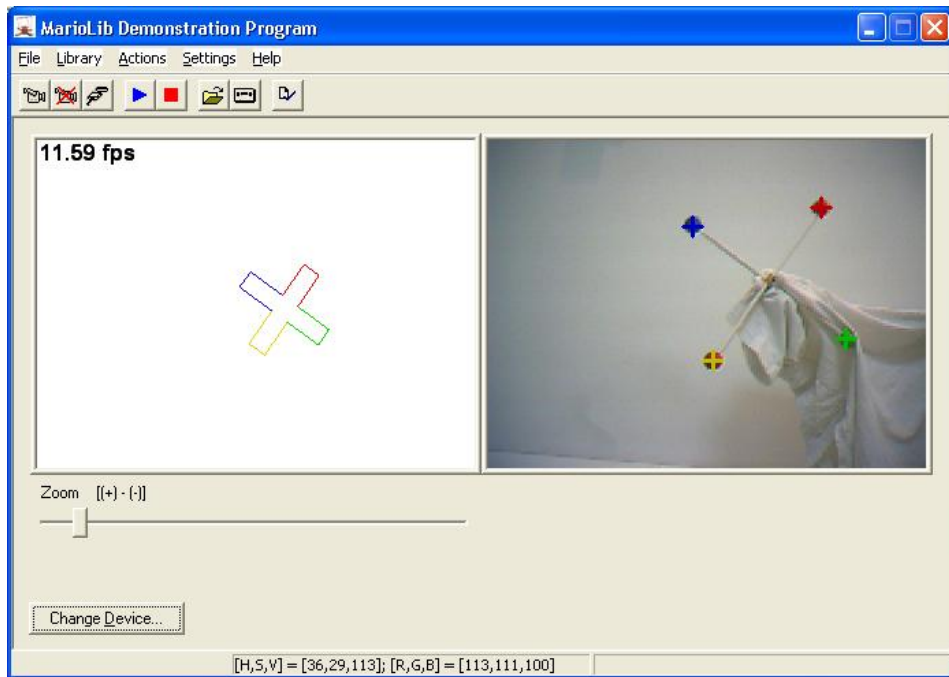
5. RESULTS AND CONCLUSIONS

5.1. Results

The object recognition is stable given that the camera settings are set to accommodate the lighting conditions (gain, exposure etc.). Otherwise, strong noise is introduced and the recognition becomes less stable.

The following are some screenshots of the application we implemented. In this application we can see on the right window the camera input (or AVI input), and the balls recognition. On the left window we can see the reconstruction of the cross.





5.2. Problems and enhancements

If a low quality WebCam is used, there can be some minor problems in the recognition of the balls. Because of the low quality of the web camera, strong green noise is present and sometimes it confuses the vision algorithm and therefore the recognition of the green ball is not stable. Changing the camera gain or using a better Web camera usually solves this problem.

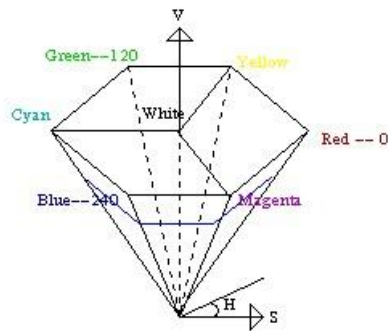
We, at first, used a Logitech's QuickCam web camera, and experienced strong green noise, which made the vision algorithm difficulties recognizing the green ball, later we moved to a better web camera: Veo's Velocity Connect, and the recognition improved dramatically.

6. REFERENCES

1. Alfred Bruckstein, Robert Holt, Thomas Huang, Arun Netravali
Optimum Fiducials under Weak Perspective Projection
International Journal of Computer Vision, 1999
2. B.K.P. Horn , Projective Geometry Considered Harmful, 1999
3. J.D.Foley, A.Van-Dam, S.K.Feiner and J.F.Hughes, Computer Graphics -
Principles and Practice
4. B. K. P. Horn, Robot Vision, 1986.
5. A. K. Jain, Fundamentals of Digital Image Processing, 1989
6. James Bruce, Tucker Balch, Manuela Veloso, Fast and Inexpensive Color
Image Segmentation for Interactive Robots, School of Computer Science
Carnegie Mellon University
7. James Bruce, Realtime Machine Vision Perception and Prediction, 2000

8. APPENDIX A: THE HSV COLOR MODEL

The HSV (Hue, Saturation, Value) color model, a cone, is shown in the figure. This is one of the perceptual color spaces and was designed to mimic the way humans perceive color. The HSV color cone defines a color by hue, saturation, and value (brightness). The value or brightness of the color varies from zero to one along the axis, and the saturation of the color varies as the radial distance from the center axis. The hue is represented as an angle, with $H = 0$ degrees being red.



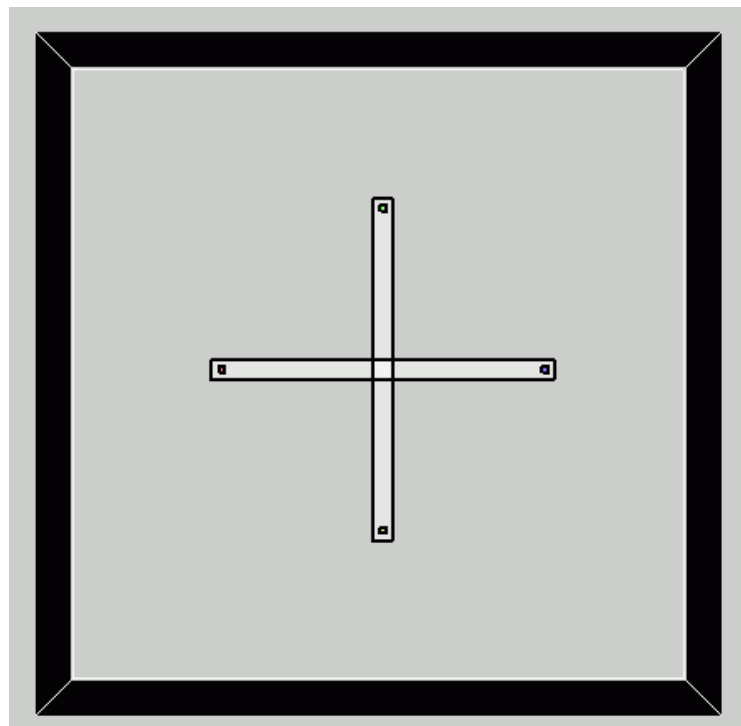
9. APPENDIX B: MATLAB IMPLEMENTATION OF THE RECONSTRUCTION

The main function (`msim_exp`) gets the parameters of the transformation which are:

1. Name - any string
2. Rot angle Z
3. Rot angle X
4. Rot angle Y
5. Displacement X
6. Displacement Y
7. Displacement Z

The procedure:

We created a 3D cross to illustrate our real cross. This cross initial location look:



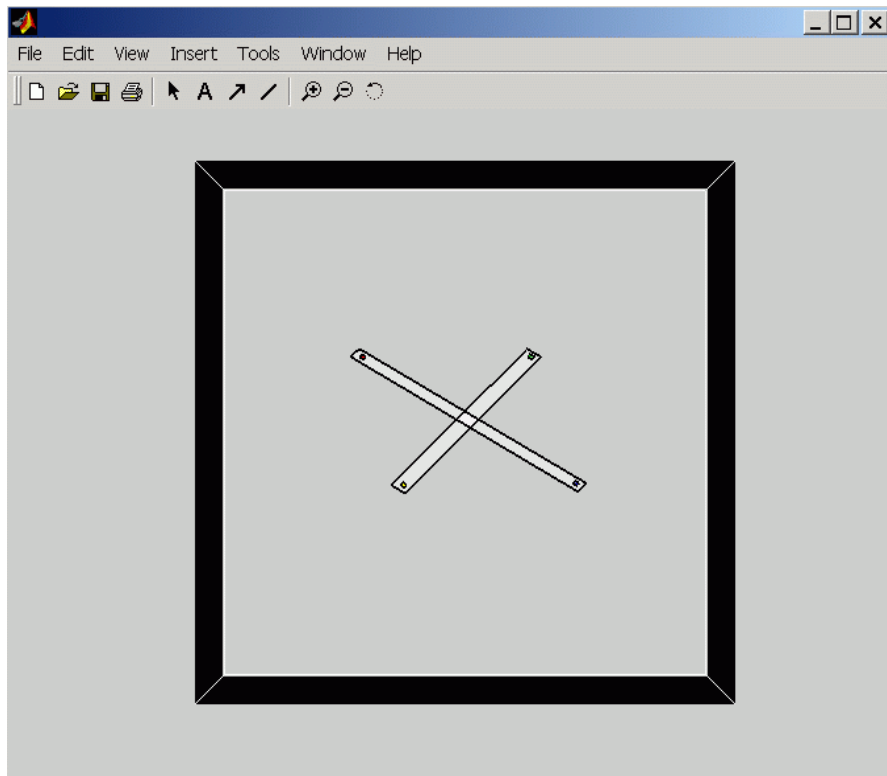
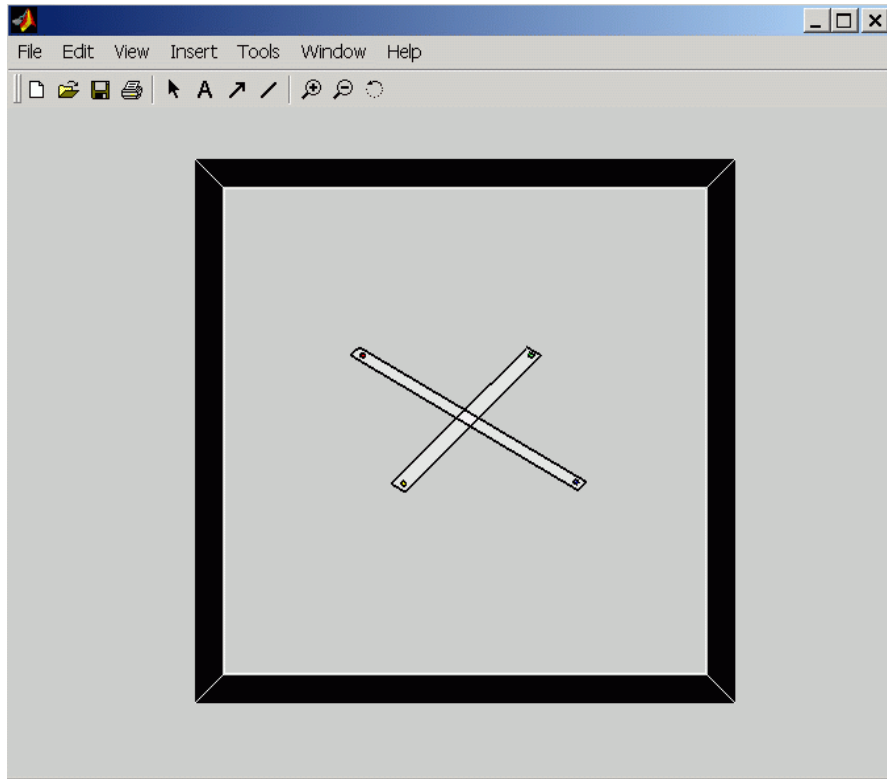
1. From properties above we build the 4X4 matrix that defines the transformation.
2. We let matlab calculate the new position of the 4 corners of the cross (using the original 4 points and the 4X4 matrix).
3. We take the result (4 new points of the new 2D image) and reconstruct the movement with our calculations (as described earlier in the document). We build another 4X4 matrix with our algorithm and compare it with the original one.
4. To check our results visually, we let matlab calculate the new position of the cross with our new matrix and we see if the two images (of two transformations, real and reconstructed) seem similar.

Results:

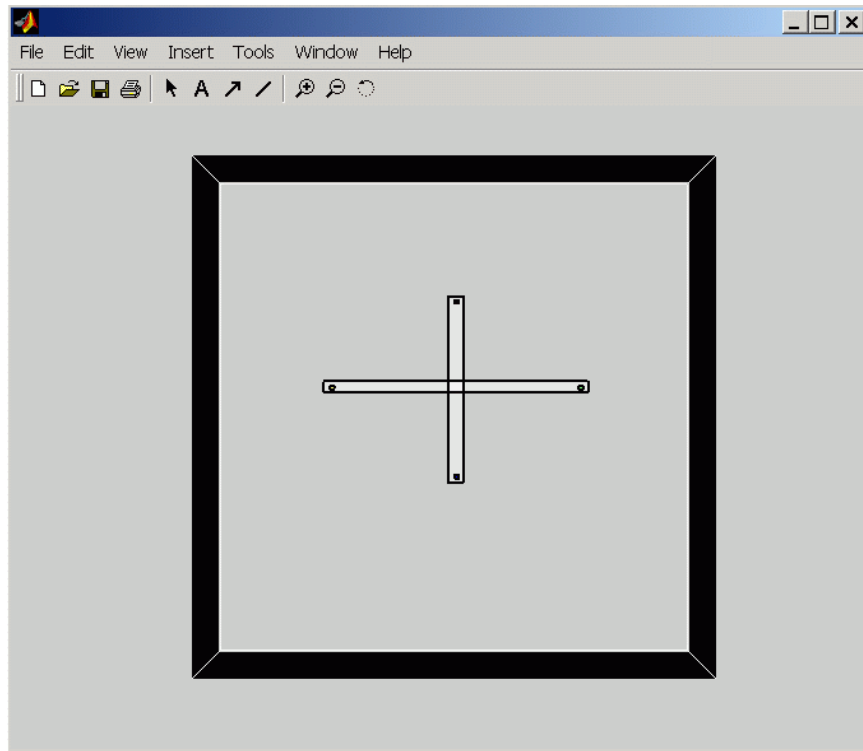
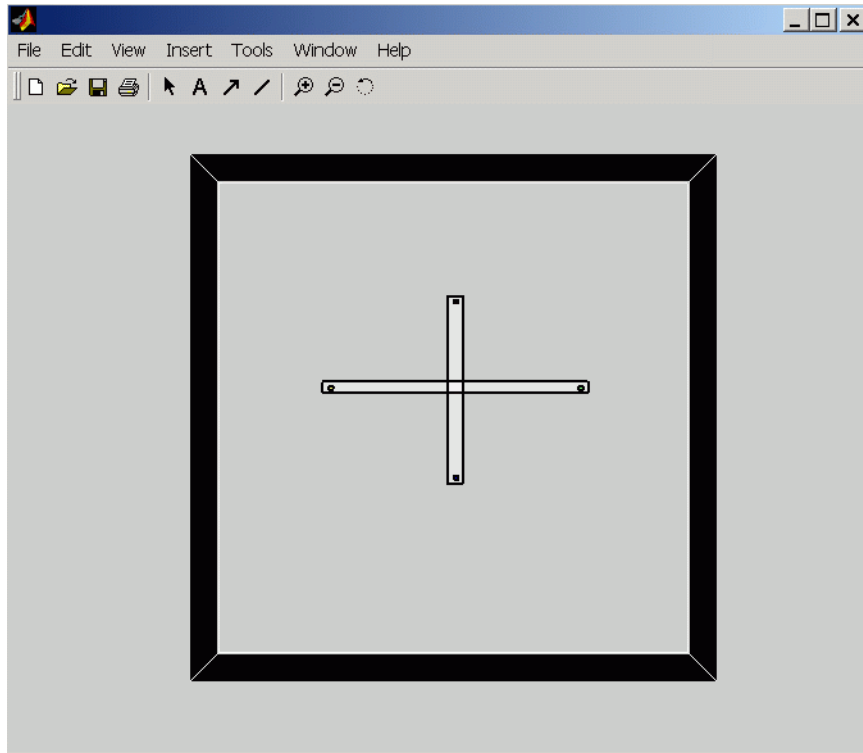
We tried the simulation on several rotations and displacements and the reconstructed matrix was very similar to the original one and in some cases the matrices were equal.

The following are some examples of the transformed cross. The first image is the real transformed cross and the second is the reconstructed transformation.

```
msim_exp('first_test', 45, 45, 20, 0, 10, 10);
```



```
msim_exp('first_test', 90, 0, 45, 0, 30, 30);
```



10. APPENDIX C: OLD ALGORITHM FOR ORIENTATION AND TRANSLATION RECONSTRUCTION

The task here is to recover the rotation 3x3 matrix and the translation 3x1 vector given the 4 correspondence points between the image plane and the object plane (note that the object is necessarily planar, so the coordinates in the object space are also 2D). See [Horn 11] for more details.

In order to achieve this, we first have to find the 3x3 matrix T , which represents a homogeneous transformation from the object plane to the image plane. T multiplied by a 3-vector $(x_o, y_o, 1)^T$ representing position in the object plane yields a 3-vector $(ku, kv, k)^T$ that represents the corresponding position in the image plane—both in homogeneous coordinates:

$$\begin{pmatrix} ku \\ kv \\ k \end{pmatrix} = \begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{pmatrix} \begin{pmatrix} x_t \\ y_t \\ 1 \end{pmatrix}$$

Given the scale factor ambiguity, we can arbitrarily pick $t_{33} = 1$ and choose

the other eight elements of T independently.

In order to recover T we solve a linear system of 8 equations:

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -y_1u_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2u_2 & -y_2u_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3u_3 & -y_3u_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4u_4 & -y_4u_4 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1v_1 & -y_1v_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2v_2 & -y_2v_2 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3v_3 & -y_3v_3 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4v_4 & -y_4v_4 \end{pmatrix} \begin{pmatrix} t_{11} \\ t_{12} \\ t_{13} \\ t_{21} \\ t_{22} \\ t_{23} \\ t_{31} \\ t_{32} \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix}$$

where (x,y) are the coordinates in the object plane and (u,v) are the coordinates in the image plane.

To recover the rotation and translation info from T , we use the concept of vanishing points.

Recovery of rotation: the vanishing point for the x-axis is just $(1, 0, 0)^T$ in the object coordinate system. Multiplying the matrix T by this vector yields the homogeneous image coordinate $(t11, t21, t31)^T$. Similarly, we get $(t12, t22, t32)^T$ from $(0, 1, 0)^T$ for the y-axis. These two correspond to image coordinates $(t11/t31, t21/t31)^T$ and $(t12/t32, t22/t32)^T$ respectively. If we connect the center of projection to these points in the image plane we obtain direction vectors parallel to:

$$x = (t11, t21, f t31)^T \quad y = (t12, t22, f t32)^T$$

We can divide these two vectors by their magnitude to obtain unit vectors \hat{x} and \hat{y} in the direction of the x- and y-axes of the object plane (expressed in the camera coordinate system). Since the z-axis — perpendicular to the object plane — has to be at right angles to any line in the object plane, we can find its direction simply by taking the cross product of the directions of the x- and y-axes found above. A rotation 3x3 matrix relating (3D) object coordinates to (3D) camera coordinates can now be constructed by adjoining the three unit column vectors in the directions of the coordinate axes:

$$R = (\hat{x}, \hat{y}, \hat{z})$$

where $\hat{x}, \hat{y}, \hat{z}$ are unit column vectors constructed from T , as described above.

Recovery of translation: the homogeneous coordinates of the origin in the object plane are obviously just $(0, 0, 1)^T$. Multiplying T by this vector yields

$(t_{13}, t_{23}, t_{33})^T$. The image of the origin of the object coordinate system then is at $(t_{13}/t_{33}, t_{23}/t_{33})^T$. Connecting the origin to this point in the image plane ($z = f$), yields a vector parallel to $t = (t_{13}, t_{23}, f t_{33})$

So we found direction of the translational vector to the object origin directly from the last column of T . We can find the distance to the object origin from the center of projection if we can determine the magnification of a line parallel to the image plane at that distance (that is, the ratio of the length of the line in the image to the length of the line on the object). If the magnification is M (typically less than one), then the z -component of the translation vector must be f/M . We can use this value to scale the direction vector t found above.

The linear magnification factor M can be computed directly by taking the square root of the determinant of matrix T .

Once we know the magnification M we can determine the translational offset of the object origin from the camera origin by multiplying $t = (t_{13}, t_{23}, f t_{33})^T$ by $M/(f t_{33})$.

We have recovered the rotation 3×3 matrix and the translation 3×1 vector corresponding to the object position and orientation in 3D space, and can now combine the two to obtain the 4×4 matrix (using homogenous coordinates) encapsulating both the rotation and the translation.