

Gernot Hoffmann

## Video tracking, using a single calibrated camera

### Tracking objects

#### Table of Contents

1. Task description	2
2. Coordinate systems	2
3. Camera	3
4. Object	4
5. Coordinate transformations	4
6. Transformations with homogeneous coordinates	5
7. Nonlinear Collinear Equations and <i>Newton's Iteration</i>	6
8. Linearized Collinear Equations	7
9. Euler angles in base coordinates	9
10. Results	10
11. PostScript program	15
12. References	31

Go to Chapter by clicking on the line  
Go to Table of Contents by clicking on the top or bottom stripe of the page

## 1. Task description

A rectangle with known edge lengths  $2a$ ,  $2b$  and corners  $P_1$  to  $P_4$  moves in front of a calibrated camera. The corners are identified in the image, delivering coordinates  $f_1, g_1$  to  $f_4, g_4$ . The task is to calculate the position and the three Euler angles of the rectangle, primarily in camera coordinates and, if necessary, in a ground based system. The method is valid for other known objects, if three or more points are identified. The rectangle can be interpreted as a bounding quadrilateral of a face, moving in front of a monitor.

## 2. Coordinate systems

All coordinate systems are right-handed. The axes are parallel as long as the Euler angles are zero. All axes are using the same scale and the same units, for instance meters, centimeters or millimeters. The orientation of the axes is different in previous publications by the author, e.g. [4]-[7]. The optical axis of the camera is now the third axis, which is more convenient for applications of homogeneous coordinates (chapter 6).

**2.1**  $x, y, z$  is the global or base system, for instance fixed to a monitor. When it is fixed in the center of the monitor,  $x$  points to the left,  $y$  upward and  $z$  perpendicular out of the monitor in forward direction, viewing from behind the monitor.

**2.2**  $u, v, w$  is the camera system. It is centered at  $\mathbf{c}=(c_x, c_y, c_z)^T$ . The superscript T means *transposed*. Vectors are primarily defined as columns.  $u$  points to the left,  $v$  upward and  $w$  on the optical axes in view direction.

The camera can be rotated by three angles relativ to the base system  $x, y, z$ . All angles count positive about the respective axis in the sense of a right screw. See illustration Fig.2 next page.

1. Azimuth  $\beta$  about the vertical axis
2. Elevation  $\alpha$  about the new lateral axis
3. Roll angle  $\gamma$  about the optical axis

If the camera is fixed at the upper edge of the monitor in center position, then we have  $\mathbf{c}=(0, c_y, 0)^T$  and the angles can be considered as small errors of alignment. But one may have as well the situation that the camera is on purpose somewhat tilted by an elevation angle in order to improve the field of view.

**2.3**  $f, g, h$  is the coordinate system for the image plane, as well represented by three axes.

The intrinsic image plane  $f, g$  in our pin hole camera model is placed perpendicularly to the optical axis, shifted by distance  $d$  from the center of projection, but in reality backward.  $d$  is the camera constant or *Kammerkonstante* – not a *calibrated focal length*, because a pin hole camera does not have a focal length. The third coordinate  $h$  completes the 3D-system.

**2.4**  $k, l, m$  is the object or body system.

It is centered at  $\mathbf{b}=(b_u, b_v, b_w)^T$  in the camera system and it can be rotated relative to the camera system by three angles:

1. Yaw angle  $\psi$  about the vertical axis
2. Pitch angle  $\theta$  about the new lateral axis
3. Roll angle  $\phi$  about the new longitudinal axis

The observed object is defined in this system. General object points  $P_j$  may be used, but later a rectangle will be defined by  $P_1=(a, b, 0)$ ,  $P_2=(-a, b, 0)$ ,  $P_3=(-a, -b, 0)$ ,  $P_4=(a, -b, 0)$ . The adjoint vectors are  $\mathbf{k}_1=(a, b, 0)^T$  etc.

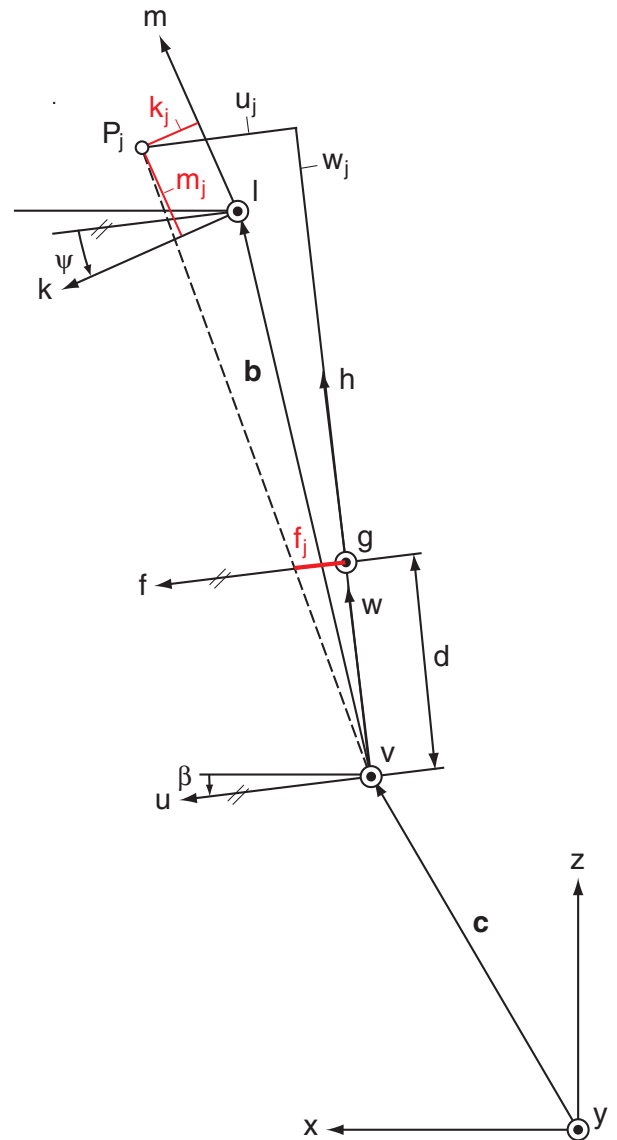


Fig.1 Coordinate systems

### 3. Kamera

The derivation of the rotation matrices is explained by local coordinates  $\mathbf{x}_1$  to  $\mathbf{x}_4$ .

$\mathbf{x}_1$  is parallel to the base system  $\mathbf{x}=(x,y,z)^T$ . First rotation by  $\beta$  into  $\mathbf{x}_2$ , then by  $\alpha$  into  $\mathbf{x}_3$  and finally by  $\gamma$  into  $\mathbf{x}_4$ .

The last rotation is not visualized by a mechanical frame.  $\mathbf{x}_4$  is identical with  $\mathbf{u}=(u,v,w)^T$ .

1. Azimuth  $\beta$  about the vertical axis  $y_1=y_2$
2. Elevation  $\alpha$  about the lateral axis  $x_2=x_3$
3. Roll angle  $\gamma$  about the optical axis  $z_3=z_4$

A rotation matrix transforms the coordinates of the same point between two coordinate systems.

Translations are taken into account by additional terms. A rotation and a translation can be combined in one homogeneous matrix (chapter 6).

$$\mathbf{x}_4 = \mathbf{Z}_{43} \mathbf{X}_{32} \mathbf{Y}_{21} \mathbf{x}_1 = \mathbf{C}_{41} \mathbf{x}_1$$

$$\mathbf{Z}_{43} = \begin{bmatrix} \cos \gamma & \sin \gamma & 0 \\ -\sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{X}_{32} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha \\ 0 & -\sin \alpha & \cos \alpha \end{bmatrix}$$

$$\mathbf{Y}_{21} = \begin{bmatrix} \cos \beta & 0 & -\sin \beta \\ 0 & 1 & 0 \\ \sin \beta & 0 & \cos \beta \end{bmatrix}$$

Rotation matrices contain always the well-known structure with two sine and two cosine functions. Where to put the minus sign, can be checked easily, here for  $\mathbf{Z}_{43}$ :

For small and positive angles  $\gamma$  the following relation is found, which can be interpreted easily for positive values  $x_3 > 0$  and  $y_3 > 0$ .

$$x_4 = x_3 \cos \gamma + y_3 \sin \gamma$$

$$\gamma \rightarrow 0$$

$$x_4 \rightarrow x_3 + y_3 \gamma > x_3$$

One can see in Fig.3,  $x_4$  is a little larger than  $x_3$ , therefore we have the positive sign. Note that the sign arrangement in the matrix looks different for a rotation about the *second* axis.

The complete rotation matrix  $\mathbf{C}_{41}$  is found by concatenating (multiplying) all matrices:

$$\mathbf{C}_{41} = \mathbf{Z}_{43} \mathbf{X}_{32} \mathbf{Y}_{21} = \begin{bmatrix} +\cos \beta \cos \gamma + \sin \alpha \sin \beta \sin \gamma & +\cos \alpha \sin \gamma & -\sin \beta \cos \gamma + \sin \alpha \cos \beta \sin \gamma \\ -\cos \beta \sin \gamma + \sin \alpha \sin \beta \cos \gamma & +\cos \alpha \cos \gamma & +\sin \beta \sin \gamma + \sin \alpha \cos \beta \cos \gamma \\ +\cos \alpha \sin \beta & -\sin \alpha & +\cos \alpha \cos \beta \end{bmatrix}$$

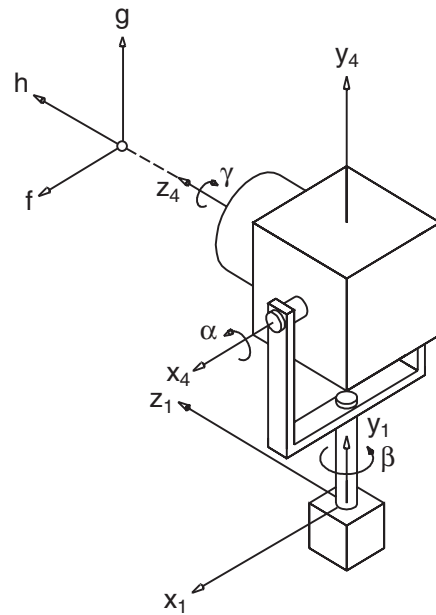


Fig.2 Camera coordinates

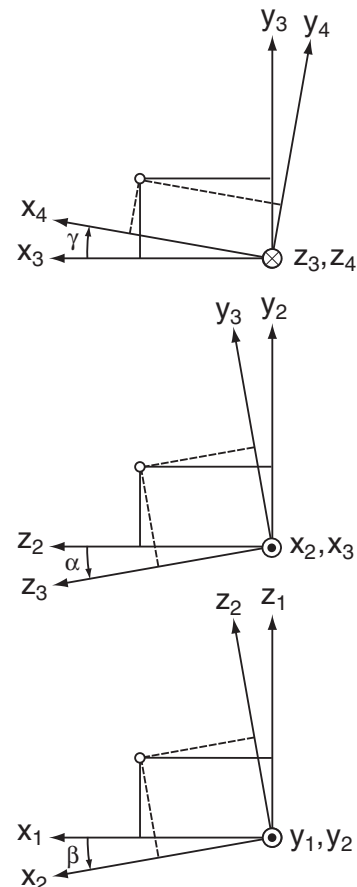


Fig.3 Sequence of camera rotations

#### 4. Object

Again, the derivation of the rotation matrices is explained by local coordinates  $\mathbf{x}_1$  to  $\mathbf{x}_4$ .

$\mathbf{x}_1$  is parallel to the camera system  $\mathbf{u}$ . First rotation by  $\psi$  into  $\mathbf{x}_2$ , then by  $\theta$  into  $\mathbf{x}_3$  and finally by  $\phi$  into  $\mathbf{x}_4$ .

$\mathbf{x}_4$  is identical with the object or body system  $\mathbf{k}=(k,l,m)^T$ .

1. Yaw angle  $\psi$  about the vertical axis  $y_1=y_2$
2. Pitch angle  $\theta$  about the lateral axis  $x_2=x_3$
3. Roll angle  $\phi$  about the long axis  $z_3=z_4$

This sequence of Euler angles is the same as for the camera. It is only necessary to exchange the names of the angles.

$\alpha \rightarrow \theta$

$\beta \rightarrow \psi$

$\gamma \rightarrow \phi$

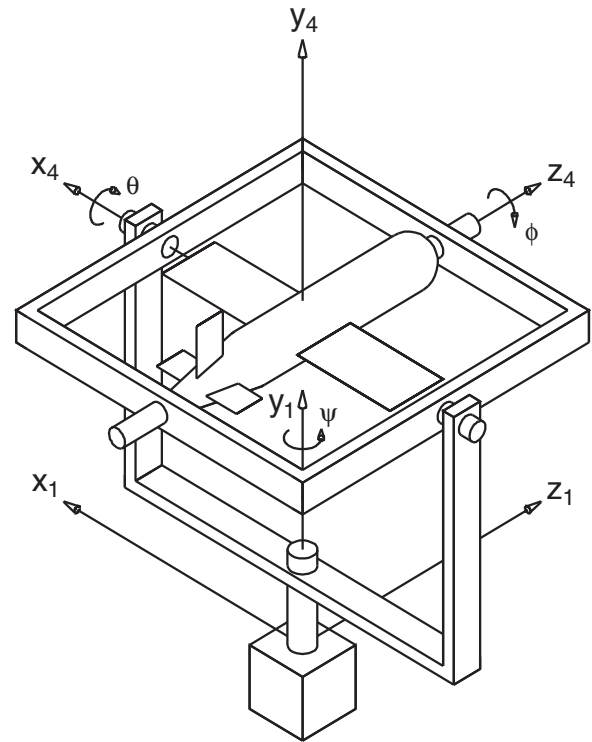


Fig.4 Object coordinates

This final rotation matrix is called **A** because it has been used often for aircraft.

$$\mathbf{A}_{41} = \mathbf{Z}_{43} \mathbf{X}_{32} \mathbf{Y}_{21} = \begin{bmatrix} +\cos\psi \cos\phi + \sin\theta \sin\psi \sin\phi & +\cos\theta \sin\phi & -\sin\psi \cos\phi + \sin\theta \cos\psi \sin\phi \\ -\cos\psi \sin\phi + \sin\theta \sin\psi \cos\phi & +\cos\theta \cos\phi & +\sin\psi \sin\phi + \sin\theta \cos\psi \cos\phi \\ +\cos\theta \sin\psi & -\sin\theta & +\cos\theta \cos\psi \end{bmatrix}$$

#### 5. Coordinate transformations

The same point P with coordinates  $\mathbf{k}=(k,l,m)^T$  is represented by coordinate sets in different coordinate systems.

Rotation matrices are orthonormal. The inverse matrix is simply found by transposing. The inverse matrix  $\mathbf{A}_{14} = \mathbf{A}_{41}^T$  is named **B** because the object system is sometimes called body system.

A matrix can be replaced by a column of row matrices. For instance,  $\mathbf{b}_1^T$  is the first row of matrix **B**. Hopefully this is understandable in the context. Otherwise one could write  $\mathbf{b}^1$  for the first row.

Object points  $\mathbf{k}$  are perspectively mapped to image points f,g, using the rule of proportion, see Fig.1 .

$$\mathbf{C} = \mathbf{C}_{14} = \mathbf{C}_{41}^T = \begin{bmatrix} \mathbf{c}_1^T \\ \mathbf{c}_2^T \\ \mathbf{c}_3^T \end{bmatrix}$$

$$\mathbf{B} = \mathbf{A}_{14} = \mathbf{A}_{41}^T = \begin{bmatrix} \mathbf{b}_1^T \\ \mathbf{b}_2^T \\ \mathbf{b}_3^T \end{bmatrix}$$

$$\mathbf{u} = \mathbf{B}\mathbf{k} + \mathbf{b}$$

$$\mathbf{x} = \mathbf{C}\mathbf{u} + \mathbf{c}$$

$$\mathbf{x} = \mathbf{C}(\mathbf{B}\mathbf{k} + \mathbf{b}) + \mathbf{c}$$

$$f/d = u/w$$

$$g/d = v/w$$

## 6. Transformation in homogeneous coordinates

The sequence *first rotation, then translation* can be combined in one matrix, with the special benefit that the perspective transform is contained as well.

The coordinate set is extended by a fourth variable  $q$ . After the execution of the matrix multiplication, all variables are divided by the fourth (the first three are divided and the fourth is set to one).

Cascaded transformations, each one possibly a rotation and/or a scaling and/or a translation are executed by matrix multiplication. A pure translation contains the identity matrix  $\mathbf{I}$  instead of a rotation matrix  $\mathbf{C}$ , see below.

It is not necessary to execute the division after translations or rotations, but the division is essential for the perspective transform by means of the matrix  $\mathbf{P}_h$ .

$$\mathbf{x}_h = \begin{bmatrix} x \\ y \\ z \\ q \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_x \\ c_{21} & c_{22} & c_{23} & c_y \\ c_{31} & c_{32} & c_{33} & c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{C} & \mathbf{c} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ 1 \end{bmatrix} = \mathbf{C}_h \mathbf{u}_h$$

$$x := x/q; \quad y := y/q; \quad z := z/q$$

$$\mathbf{u}_h = \begin{bmatrix} u \\ v \\ w \\ q \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_u \\ b_{21} & b_{22} & b_{23} & b_v \\ b_{31} & b_{32} & b_{33} & b_w \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} k \\ l \\ m \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{B} & \mathbf{b} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{k} \\ 1 \end{bmatrix} = \mathbf{B}_h \mathbf{k}_h$$

$$u := u/q; \quad v := v/q; \quad w := w/q$$

$$\mathbf{f}_h = \begin{bmatrix} f \\ g \\ h \\ q \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ 1 \end{bmatrix} = \mathbf{P}_h \mathbf{u}_h$$

$$f := f/q; \quad g := g/q; \quad h := h/q$$

$$\mathbf{x}_h = \mathbf{C}_h \mathbf{u}_h$$

$$\mathbf{u}_h = \mathbf{B}_h \mathbf{k}_h$$

$$\mathbf{x}_h = \mathbf{C}_h \mathbf{B}_h \mathbf{k}_h$$

$$\mathbf{f}_h = \mathbf{P}_h \mathbf{B}_h \mathbf{k}_h$$

$$f = du/w$$

$$g = dv/w$$

Such a homogeneous operator cannot be inverted by using the inverse matrix. Solving the first equation for  $\mathbf{u}$ , one has to apply first a translation and then a rotation. Each operation can be represented by a homogeneous matrix. Multiplying results in  $\mathbf{C}_h^{-1}$  as shown below.

$$\mathbf{x} = \mathbf{C}\mathbf{u} + \mathbf{c}$$

$$\mathbf{u} = \mathbf{C}^{-1}(\mathbf{x} - \mathbf{c}) \quad \text{with } \mathbf{C}^{-1} = \mathbf{C}^T$$

$$\mathbf{u}_h = \begin{bmatrix} \mathbf{C}^{-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\mathbf{c} \\ \mathbf{0}^T & 1 \end{bmatrix} \mathbf{x}_h \quad \text{with 3x3- identity matrix } \mathbf{I}$$

$$\mathbf{u}_h = \begin{bmatrix} \mathbf{C}^{-1} & -\mathbf{C}^{-1}\mathbf{c} \\ \mathbf{0}^T & 1 \end{bmatrix} \mathbf{x}_h = \mathbf{C}_h^{-1} \mathbf{x}_h$$

## 7. Nonlinear Collinear Equations and *Newton's Iteration*

The method as described in this chapter is based on reference [2]. An arbitrary point  $P_j$  with object coordinates  $\mathbf{k}_j = (k, l, m)^T$  is perspectively mapped to the image plane as follows. Note: vectors  $\mathbf{b}_i^T$  etc. are rows.

$$f_j = d \frac{u_j}{w_j} = d \frac{b_u + \mathbf{b}_1^T \mathbf{k}_j}{b_w + \mathbf{b}_3^T \mathbf{k}_j}$$

$$g_j = d \frac{v_j}{w_j} = d \frac{b_v + \mathbf{b}_2^T \mathbf{k}_j}{b_w + \mathbf{b}_3^T \mathbf{k}_j}$$

For a rectangle  $P_1, \dots, P_4$  one has four pairs of these equations. Unknown are the three translations  $b_u, b_v, b_w$  and the three Euler angles  $\psi, \theta, \phi$ .

These variables can be combined in one vector of unknowns:

$$\mathbf{p} = (b_u, b_v, b_w, \psi, \theta, \phi)^T$$

Because of the many trigonometric functions in  $\mathbf{b}_1^T, \mathbf{b}_2^T, \mathbf{b}_3^T$  (the rows of the rotation matrix  $\mathbf{B}$ ), the system of equations is strongly nonlinear. The division itself is already a nonlinear operation.

A solution is possible by *Newton's Iteration*:

$$f_j = d \frac{u_j}{w_j} = d \frac{b_u + \mathbf{b}_1^T \mathbf{k}_j}{b_w + \mathbf{b}_3^T \mathbf{k}_j}$$

using  $u_j, v_j, w_j$  like here

$$g_j = \dots$$

and in  $g_j$  as above delivers

$$F_1 = f_1 / d - u_1 / w_1 = 0$$

8 Collinear Equations

...

$$F_4 = f_4 / d - u_4 / w_4 = 0$$

$$F_5 = g_1 / d - v_1 / w_1 = 0$$

...

$$F_8 = g_4 / d - v_4 / w_4 = 0$$

$$\mathbf{F} = \mathbf{0}$$

is expected for the solution  $\mathbf{p}$

$$\mathbf{p} = \mathbf{p}_i$$

start parameters

$$\mathbf{F} = \mathbf{F}_i + \mathbf{J}_i \mathbf{dp} = \mathbf{0}$$

$\mathbf{F}_i = \mathbf{F}(\mathbf{p}_i)$ , correction by  $\mathbf{dp}$

$$\mathbf{J}_i = \begin{bmatrix} \partial F_1 / \partial p_1 & \dots & \partial F_1 / \partial p_6 \\ \dots & \dots & \dots \\ \partial F_8 / \partial p_1 & \dots & \partial F_8 / \partial p_6 \end{bmatrix}_i$$

*Jacobi matrix* at  $\mathbf{p}_i$

$$\mathbf{J}_i \mathbf{dp} = -\mathbf{F}_i$$

8 equations, 6 unknowns

$$\mathbf{J}_i^T \mathbf{J}_i \mathbf{dp} = -\mathbf{J}_i^T \mathbf{F}_i$$

*Gaussian normal transform*

$$\mathbf{M} \mathbf{dp} = \mathbf{m}$$

6 equations, symmetric matrix

$$\mathbf{p}_i = \mathbf{p}_i + \mathbf{dp}$$

improved parameter vector

At first one puts up the eight *Collinear Equations*  $F_1=0 \dots F_8=0$  using the complete quotients. The equations are zero for correct parameters  $\mathbf{p}$ . The parameters are changed with *Newton's method*. This requires the so-called *Jacobi matrix*  $\mathbf{J}$  which contains all partial derivatives. These can be approximated by numerical differentiation. This leads to a linearization of the function  $\mathbf{F}$  in a six-dimensional space.

One expects  $\mathbf{F}=\mathbf{0}$  for  $\mathbf{p}=\mathbf{p}_i+\mathbf{dp}$ . The system has 8 linear equations for 6 unknowns. A least squares compromise is found by the so-called *Gaussian normal transform* which delivers 6 linear equations for 6 unknowns.  $\mathbf{M}$  is symmetric, therefore the equations can be solved by the method of *Cholesky* [1] or by the *Gaussian equation solver* [1], as shown in chapter 11. *Cholesky* is somewhat faster.

The process has to be repeated until a norm of  $\mathbf{F}$  is sufficiently small. The costs are rather high, especially for the partial derivatives. It would be necessary to represent the trigonometric functions by tables.

Unfortunately it is not guaranteed that *Newton's Iteration* converges. Therefore it is in the moment not recommended to apply this algorithm.

## 8. Linearized Collinear Equations

The transformation from object to camera coordinates is started by this equation:

$$\mathbf{u} = \mathbf{B}\mathbf{k} + \mathbf{b}$$

For small angles  $\alpha = \psi, \theta, \phi$  one may replace  $\cos\alpha = 1$ ,  $\sin\alpha = \alpha$  and ignore products  $\psi\phi$  etc. Thus, after multiplying with the common denominator, these linear equations are found:

$$\mathbf{u}_j = \hat{\mathbf{B}}\mathbf{k}_j + \mathbf{b}$$

$$\hat{\mathbf{B}} = \begin{bmatrix} 1 & -\phi & +\psi \\ +\phi & 1 & -\theta \\ -\psi & +\theta & 1 \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{b}}_1^T \\ \hat{\mathbf{b}}_2^T \\ \hat{\mathbf{b}}_3^T \end{bmatrix}$$

$$f_j/d = \frac{u_j}{w_j} = \frac{b_u + \hat{\mathbf{b}}_1^T \mathbf{k}_j}{b_w + \hat{\mathbf{b}}_3^T \mathbf{k}_j}$$

$$g_j/d = \frac{v_j}{w_j} = \frac{b_v + \hat{\mathbf{b}}_2^T \mathbf{k}_j}{b_w + \hat{\mathbf{b}}_3^T \mathbf{k}_j}$$

Multiplied by the denominator:

$$(f_j/d)(b_w + \hat{\mathbf{b}}_3^T \mathbf{k}_j) = b_u + \hat{\mathbf{b}}_1^T \mathbf{k}_j$$

$$(g_j/d)(b_w + \hat{\mathbf{b}}_3^T \mathbf{k}_j) = b_v + \hat{\mathbf{b}}_2^T \mathbf{k}_j$$

Abbreviation:

$$\bar{f}_j = f_j/d$$

$$\bar{g}_j = g_j/d$$

$$\bar{f}_j (b_w - \psi k_j + \theta l_j + m_j) = b_u + k_j - \phi l_j + \psi m_j$$

$$\bar{g}_j (b_w - \psi k_j + \theta l_j + m_j) = b_v + \phi k_j + l_j - \theta m_j$$

These equations are valid for arbitrary points  $P_j$ . They are linear with respect to the unknown parameters  $\mathbf{p} = (b_u, b_v, b_w, \psi, \theta, \phi)^T$ .

$b_u$	$b_v$	$b_w$	$\psi$	$\theta$	$\phi$	= RHS
-1	0	$\bar{f}_j$	$-(\bar{f}_j k_j + m_j)$	$\bar{f}_j l_j$	$l_j$	$= k_j - \bar{f}_j m_j$
0	-1	$\bar{g}_j$	$-\bar{g}_j k_j$	$(\bar{g}_j l_j + m_j)$	$-k_j$	$= l_j - \bar{g}_j m_j$

The intended rectangle is defined in the plane  $k, l$ , therefore one has generally  $m_j = 0$ :

$b_u$	$b_v$	$b_w$	$\psi$	$\theta$	$\phi$	= RHS
-1	0	$\bar{f}_j$	$-\bar{f}_j k_j$	$\bar{f}_j l_j$	$l_j$	$= k_j$
0	-1	$\bar{g}_j$	$-\bar{g}_j k_j$	$\bar{g}_j l_j$	$-k_j$	$= l_j$

RHS means *right-handed side*

Four points with  $m_j=0$  deliver these eight equations:

$b_u$	$b_v$	$b_w$	$\psi$	$\theta$	$\phi$	=	RHS
-1	0	$\bar{f}_1$	$-\bar{f}_1 k_1$	$\bar{f}_1 l_1$	$l_1$	=	$k_1$
-1	0	$\bar{f}_2$	$-\bar{f}_2 k_2$	$\bar{f}_2 l_2$	$l_2$	=	$k_2$
-1	0	$\bar{f}_3$	$-\bar{f}_3 k_3$	$\bar{f}_3 l_3$	$l_3$	=	$k_3$
-1	0	$\bar{f}_4$	$-\bar{f}_4 k_4$	$\bar{f}_4 l_4$	$l_4$	=	$k_4$
0	-1	$\bar{g}_1$	$-\bar{g}_1 k_1$	$\bar{g}_1 l_1$	$-k_1$	=	$l_1$
0	-1	$\bar{g}_2$	$-\bar{g}_2 k_2$	$\bar{g}_2 l_2$	$-k_2$	=	$l_2$
0	-1	$\bar{g}_3$	$-\bar{g}_3 k_3$	$\bar{g}_3 l_3$	$-k_3$	=	$l_3$

The rectangle has edge lengths  $2a$  and  $2b$  with this assignment for the points:

	$k_j$	$l_j$
$P_1$	+a	+b
$P_2$	-a	+b
$P_3$	-a	-b

These are the final equations:

$b_u$	$b_v$	$b_w$	$\psi$	$\theta$	$\phi$	=	RHS
-1	0	$\bar{f}_1$	$-a\bar{f}_1$	$+b\bar{f}_1$	+b	=	+a
-1	0	$\bar{f}_2$	$+a\bar{f}_2$	$+b\bar{f}_2$	+b	=	-a
-1	0	$\bar{f}_3$	$+a\bar{f}_3$	$-b\bar{f}_3$	-b	=	-a
-1	0	$\bar{f}_4$	$-a\bar{f}_4$	$-b\bar{f}_4$	-b	=	+a
0	-1	$\bar{g}_1$	$-a\bar{g}_1$	$+b\bar{g}_1$	-a	=	+b
0	-1	$\bar{g}_2$	$+a\bar{g}_2$	$+b\bar{g}_2$	+a	=	+b
0	-1	$\bar{g}_3$	$+a\bar{g}_3$	$-b\bar{g}_3$	+a	=	-b

The matrix  $\mathbf{M}$  and the right-hand side  $\mathbf{r}$  (RHS) are filled according to this scheme. Unknown is the parameter vector  $\mathbf{p}$ . For 8 equations with 6 unknowns the *Gaussian normal transform* by means of matrix  $\mathbf{M}^T$  is applied, which delivers a new system with 6 equations. This is actually a least squares approximation for the somewhat contradictory original system. The new system can be solved by *Cholesky* [6] or a by a standard *Gaussian equation solver*, as in chapter [11].

$$\mathbf{p} = (b_u, b_v, b_w, \psi, \theta, \phi)^T$$

$$\mathbf{M}\mathbf{p} = \mathbf{r}$$

$$\mathbf{M}^T\mathbf{M}\mathbf{p} = \mathbf{M}^T\mathbf{r}$$

$$\mathbf{N}\mathbf{p} = \mathbf{n}$$



## 9. Euler angles with respect to the base system.

So far it is possible to calculate the coordinates of object points P in the base system, but the Euler angles  $\psi', \theta', \phi'$  of the object with respect to this system are unknown. The information is hidden in matrix **D**.

$$\mathbf{u} = \mathbf{B}\mathbf{k} + \mathbf{b}$$

$$\mathbf{x} = \mathbf{C}\mathbf{u} + \mathbf{c}$$

$$\mathbf{x} = \mathbf{C}(\mathbf{B}\mathbf{k} + \mathbf{b}) + \mathbf{c}$$

$$\mathbf{x} = \mathbf{CB}\mathbf{k} + \mathbf{Cb} + \mathbf{c}$$

$$\mathbf{x} = \mathbf{D}\mathbf{k} + \dots$$

Matrix  $\mathbf{D} = \mathbf{CB}$  can be expressed by angles  $\psi', \theta', \phi'$ , which are meant as in Fig.4, but with respect to the base system instead of the camera system. Thus, one simply writes matrix **D** like  $\mathbf{A}_{14}$  and call it  $\mathbf{A}'_{14}$ .

$$\mathbf{A}'_{14} = \mathbf{A}'_{41}{}^T = \mathbf{D} = \begin{bmatrix} +\cos\psi' \cos\phi' + \sin\theta' \sin\psi' \sin\phi' & -\cos\psi' \sin\phi' + \sin\theta' \sin\psi' \cos\phi' & +\cos\theta' \sin\psi' \\ +\cos\theta' \sin\phi' & +\cos\theta' \cos\phi' & -\sin\theta' \\ -\sin\psi' \cos\phi' + \sin\theta' \cos\psi' \sin\phi' & +\sin\psi' \sin\phi' + \sin\theta' \cos\psi' \cos\phi' & +\cos\theta' \cos\psi' \end{bmatrix}$$

$$\mathbf{D} = (d_{ik})$$

$$\tan\psi' = \frac{d_{13}}{d_{33}}$$

$$\tan\phi' = \frac{d_{21}}{d_{22}}$$

$$\tan\theta' = \frac{(-d_{23})}{\sqrt{d_{21}^2 + d_{22}^2}}$$

The angles are calculated using the well-known four-quadrant arcustangent *atan2*. In the programming language C one gets the angle in radians in the range  $\pm\pi$ , but in PostScript in degrees in the range  $0^\circ$  to  $360^\circ$ .

angle = *atan2* (numerator, denominator)

The whole calculation is obsolete if the camera is not rotated,  $\mathbf{C} = \mathbf{I}$ .

## 10. Results

It is not clear whether the linearized version is suitable for practical applications, for instance for face tracking in front of a monitor.

This has been tested by a PostScript program (chapter 11). Only the transformation between the object system and the camera system is considered. The transformation for the base system is straightforward and therefore it was not tested.

**Page0** shows the whole test environment. The image top right is the camera image plane. Cyan dots, connected by a cyan quadrilateral, represent the image of the object rectangle.

Red dots represent the corners if the perspective transform were executed with *calculated* values for the position and the Euler angles.

The numerical values, true and calculated, are shown by numbers, as well are the matrices.

Very important are the so-called residuals and the determinant:

$$\mathbf{R}_M = \mathbf{M} \mathbf{p} - \mathbf{r}$$

$$\mathbf{R}_N = \mathbf{N} \mathbf{p} - \mathbf{n}$$

$$\mathbf{D} = \det(\mathbf{N})$$

Residuals should be zero.  $\mathbf{R}_M = \text{resM}$  cannot be zero because eight measured values for six parameters are always to a certain degree contradictory. But this effect is even stronger because of the linearization.  $\mathbf{R}_N = \text{resN}$  indicates the quality of the numerical solution. The determinant of  $\mathbf{N}$  is as well an important indicator for proper numerical handling.

All geometrical dimensions are given by millimeters. This results in a somewhat unbalanced matrix  $\mathbf{M}$ . The problem is solved by multiplying the edge lengths  $a, b$  by a scale factor  $S_c = 0.005$ . After solving the equations, the components  $b_u, b_v, b_w$  have to be divided by the same factor. The choice of the scale factor is not critical. Trial and error, observing all test data on **Page0**, should lead to a reasonable result.

PostScript itself uses in Adobe applications (here InDesign and Acrobat) single precision float, whereas Quite PSAlter [3] seems to work with double precision float, which leads to almost zero residuals  $\mathbf{R}_N$ .

So far, judging by the images on the next pages, the result looks promising. The linearized equations are probably a good starting point for *Newton's Iteration*.

Even the execution speed is surprisingly high. PostScript is slow because it is an interpreter, opposed to compiling languages. Furtheron, the implementation uses many named variables instead of immediate stack operations.

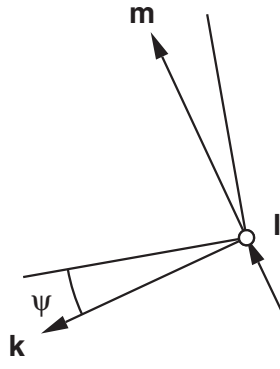
As a speed test, in procedure *Page0* the procedure *Mapping* was repeated 5000 times. This requires 35 s, which is equivalent to about 140 executions per second. An implementation in C would be much faster.

alf bet gam  
 0.0000  
 10.0000  
 0.0000

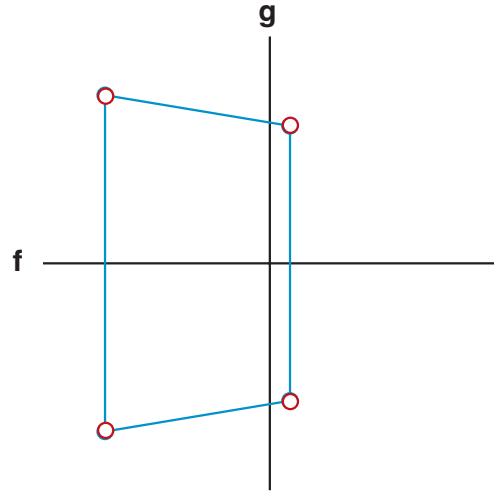
cx cy cz  
 15.0000  
 0.0000  
 40.0000

d  
 30.0000

a b  
 70.0000  
 120.0000



Sc  
 0.0050



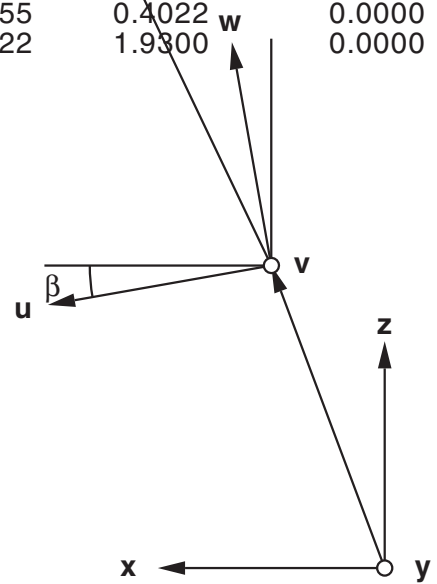
psi the phi (true)	○ psi the phi (calc)
15.0000	14.1223
0.0000	-0.0000
0.0000	0.0000

bu bv bw (true)	○ bu bv bw (calc)
50.0000	50.7511
0.0000	-0.0000
180.0000	181.2513

M86						r81
-1.0000	0.0000	0.7265	-0.2543	0.4359	0.6000	0.3500
-1.0000	0.0000	-0.0889	-0.0311	-0.0533	0.6000	-0.3500
-1.0000	0.0000	-0.0889	-0.0311	0.0533	-0.6000	-0.3500
-1.0000	0.0000	0.7265	-0.2543	-0.4359	-0.6000	0.3500
0.0000	-1.0000	0.7413	-0.2594	0.4448	-0.3500	0.6000
0.0000	-1.0000	0.6057	0.2120	0.3634	0.3500	0.6000
0.0000	-1.0000	-0.6057	-0.2120	0.3634	0.3500	-0.6000
0.0000	-1.0000	-0.7413	0.2594	0.4448	-0.3500	-0.6000

N66						n61
4.0000	0.0000	-1.2753	0.5708	0.0000	0.0000	0.0000
0.0000	4.0000	0.0000	0.0000	-1.6164	0.0000	0.0000
-1.2753	0.0000	2.9043	-0.4918	0.0000	0.0000	2.1872
0.5708	0.0000	-0.4918	0.3558	0.0000	0.0000	-0.2132
0.0000	-1.6164	0.0000	0.0000	1.0455	0.4022	0.0000
0.0000	0.0000	0.0000	0.0000	0.4022	1.9300	0.0000

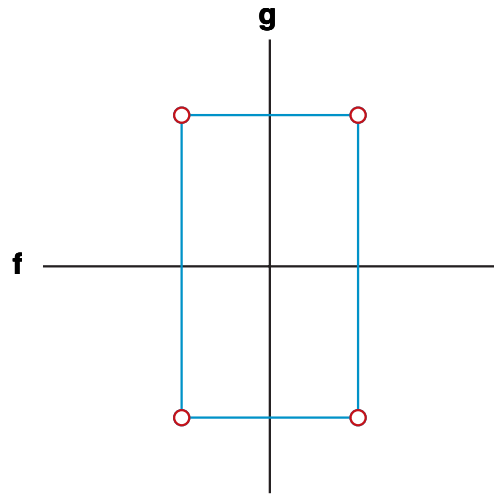
resM	resN
-0.0080	0.0000000298
0.0080	0.0000000000
0.0080	0.0000000000
-0.0080	-0.0000000298
0.0078	0.0000000000
0.0012	-0.0000000000
-0.0012	
-0.0078	Det N66
	5.6140880483



10.2 Page1

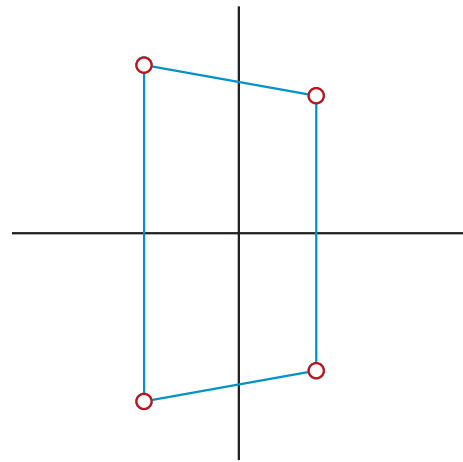
psi the phi (true)	○ psi the phi (calc)
0.0000	0.0000
0.0000	0.0000
0.0000	0.0000

bu bv bw (true)	○ bu bv bw (calc)
0.0000	0.0000
0.0000	0.0000
180.0000	180.0000



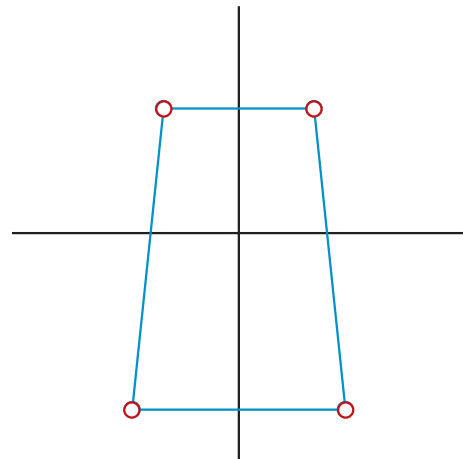
psi the phi (true)	○ psi the phi (calc)
15.0000	14.9553
0.0000	-0.0000
0.0000	0.0000

bu bv bw (true)	○ bu bv bw (calc)
0.0000	0.0000
0.0000	-0.0000
180.0000	181.5301



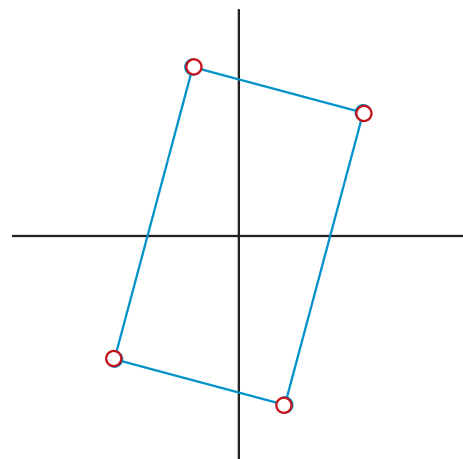
psi the phi (true)	○ psi the phi (calc)
0.0000	0.0000
15.0000	15.2126
0.0000	0.0000

bu bv bw (true)	○ bu bv bw (calc)
0.0000	0.0000
0.0000	-0.0000
180.0000	184.6528



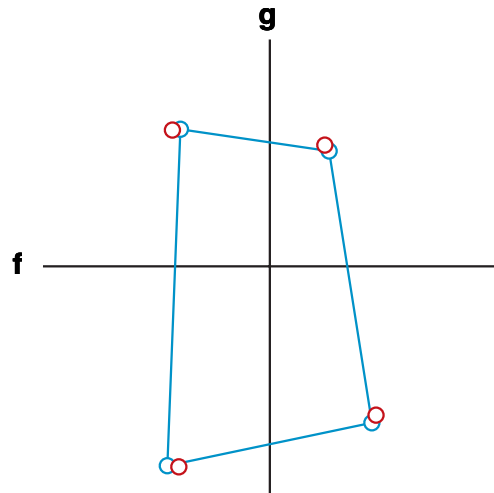
psi the phi (true)	○ psi the phi (calc)
0.0000	0.0000
0.0000	0.0000
15.0000	15.3524

bu bv bw (true)	○ bu bv bw (calc)
0.0000	-0.0000
0.0000	-0.0000
180.0000	186.3497

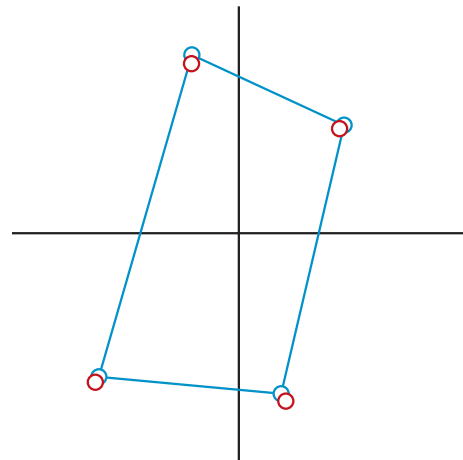


10.3 Page2

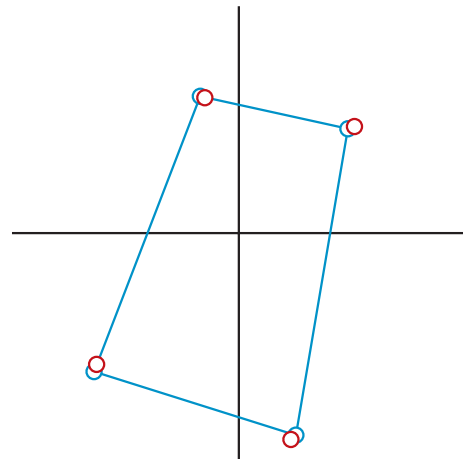
psi the phi (true)	○ psi the phi (calc)
15.0000	14.3828
15.0000	13.7863
0.0000	-2.9624
bu bv bw (true)	○ bu bv bw (calc)
0.0000	0.3680
0.0000	-1.4189
180.0000	185.7137



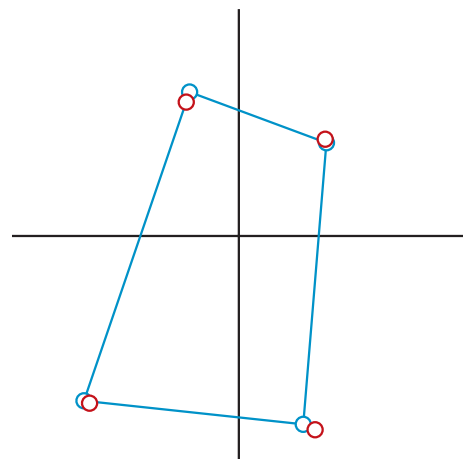
psi the phi (true)	○ psi the phi (calc)
15.0000	14.8882
0.0000	3.7522
15.0000	15.0888
bu bv bw (true)	○ bu bv bw (calc)
0.0000	0.1205
0.0000	-0.3389
180.0000	187.8992



psi the phi (true)	○ psi the phi (calc)
0.0000	-3.8216
15.0000	15.1428
15.0000	15.6125
bu bv bw (true)	○ bu bv bw (calc)
0.0000	-0.0950
0.0000	-0.1227
180.0000	191.1313

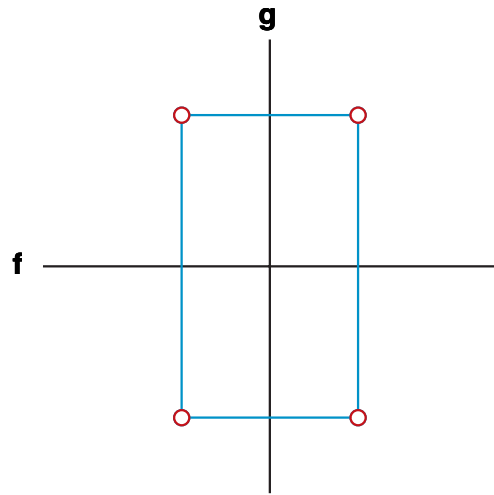


psi the phi (true)	○ psi the phi (calc)
15.0000	10.2390
15.0000	18.1650
15.0000	12.3200
bu bv bw (true)	○ bu bv bw (calc)
0.0000	0.6935
0.0000	-0.7584
180.0000	191.4585

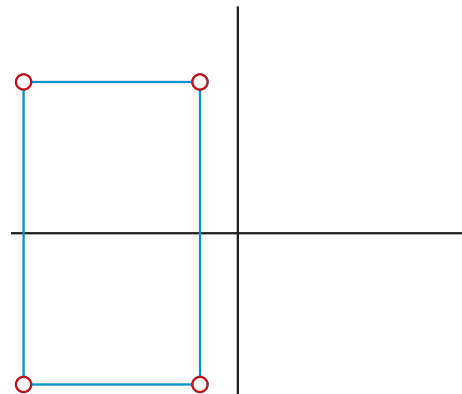


10.4 Page3

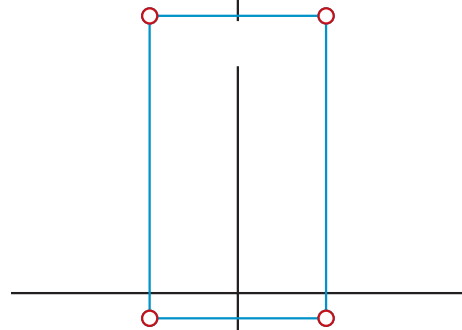
psi the phi (true)	○ psi the phi (calc)
0.0000	0.0000
0.0000	0.0000
0.0000	0.0000
bu bv bw (true)	○ bu bv bw (calc)
0.0000	0.0000
0.0000	0.0000
180.0000	180.0000



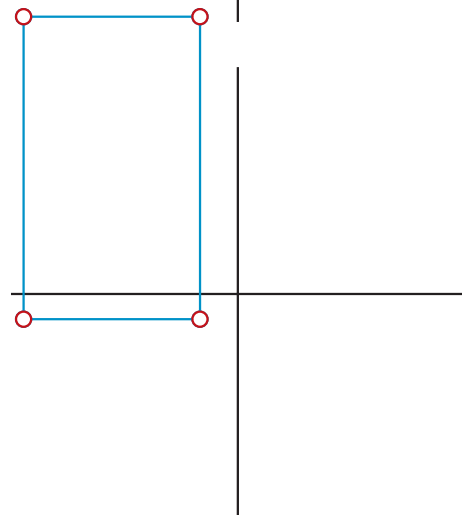
psi the phi (true)	○ psi the phi (calc)
0.0000	-0.0000
0.0000	0.0000
0.0000	0.0000
bu bv bw (true)	○ bu bv bw (calc)
100.0000	100.0000
0.0000	0.0000
180.0000	180.0000



psi the phi (true)	○ psi the phi (calc)
0.0000	0.0000
0.0000	-0.0000
0.0000	0.0000
bu bv bw (true)	○ bu bv bw (calc)
0.0000	0.0000
100.0000	100.0000
180.0000	180.0000



psi the phi (true)	○ psi the phi (calc)
0.0000	-0.0000
0.0000	0.0000
0.0000	-0.0000
bu bv bw (true)	○ bu bv bw (calc)
100.0000	100.0000
100.0000	100.0000
180.0000	179.9999



## 11. PostScript program

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 595 842
%%Creator: Gernot Hoffmann
%%Title: PS-1Kamera
%%CreationDate: May 30 / 2012
% Object tracking

/mm {2.834646 mul} def % convert mm into PostScript units
/mi {2.834646 div} def % inverse function
/gr {180 mul 3.141592 div} def % convert rad into degree

/Sc1 [0.005] def % Scaling for numerical improvement

% Graphics coordinates in mm
/xgrf1 180 def
/ygrf1 20 def
/xgrf2 150 def
/ygrf2 240 def
/dygrf 65 def
/xtxt1 30 def
/xtxt2 64 def
/xtxt3 175 def
/ytxt1 255 def
/ytxt2 100 def
/ax 30 def
/r1 1 def
/r2 1 def
/col1 {1 0 0 0.2 setcmykcolor} def
/col2 {0 1 1 0.2 setcmykcolor} def

% Camera, angles in degree, coordinates in mm
/alf 0 def
/bet 10 def
/gam 0 def
/cx 15 def
/cy 0 def
/cz 40 def
/d 30 def

% Object, angles in degree, coordinates in mm
/psi 15 def
/the 0 def
/phi 10 def
/bu 50 def
/bv 0 def
/bw 180 def

/apo 70 def % Edge lengths a,b
/bpo 120 def
```

## /SetHelv

```
{0 0 0 1 setcmykcolor
/fh 11 mi def
/Helvetica findfont fh scalefont setfont
/tms 4 def % after decimal point
/tmi 4 def % before decimal point (used for space)
} def
```

## /Shownum

```
% Draw number by string
% Call:      x y nu Shownum
% Requires: font, fh (fontheight), tms (mantissa digits)
% Version May 15 2004 (rounding)
% Version Feb 08 2007 (dict)
% tx0 ty0 nu :on stack
% tx0      Position          not overwritten
% ty0
% nu      Input number      not overwritten  nu =+-999999
% fh      Font height       not overwritten
% tms     Mantissa number of characters      tms=0...6 Global
% tms=3 Example
% input  -23.56789  -999.99    0.4567  9999.123456
% result -23.568    -999.990   0.467   9999.123
% Postscript number to string is not well defined
% e.g. 1E-5 instead of 0.00001
% We use a straightforward BCD conversion.
% This is always affected by round-off errors
% because of 32-bit arithmetic
% Results are different, depending on the interpreter
{ 1 dict
begin
/nu  exch def
/ty0 exch def
/tx0 exch def
/tfw fh 0.6 mul def % character distance
/tna nu abs 10 tms neg exp 0.500001 mul add def
/tdec 1E5 def
/tchr 1 string def
tna 999999.1 lt % larger number replaced by #
/tmm true def % sign
{/tx0 tx0 tfw 6 mul sub def
/tz 0 def
1 1 5 % first 5 digits, no leading 0
{ pop
/tk 0 def
{ tna tdec gt {/tna tna tdec sub def /tk tk 1 add def}{exit} ifelse
} loop
tk 0 ne {/tz tz 1 add def} if
tz 0 ne
{ tx0 ty0 moveto tk tchr cvs show
} if
tz 1 eq nu 0 lt and tmm and % minus Correction 28/03/2005
{ tx0 tfw 0.7 mul sub ty0 moveto (-) show
/tmm false def
} if
/tdec tdec 0.1 mul def
/tx0 tx0 tfw add def
```



```

} for
  /tk 0 def                                % leading 0
  { tna tdec gt {/tna tna tdec sub def /tk tk 1 add def}{exit} ifelse
  } loop
    tmm nu 0 lt and                          % minus
    { tx0 tfw 0.7 mul sub ty0 moveto (-) show
    } if
    tx0 ty0 moveto tk tchr cvs show
  /tdec tdec 0.1 mul def
  /tx0 tx0 tfw add def
tms 0 gt                                    % for float
{ tx0 ty0 moveto (.) show
  /tx0 tx0 tfw 0.5 mul add def
1 1 tms
{ pop
  /tk 0 def
  { tna tdec gt {/tna tna tdec sub def /tk tk 1 add def}{exit} ifelse
  } loop
    tx0 ty0 moveto tk tchr cvs show
  /tdec tdec 0.1 mul def
  /tx0 tx0 tfw add def
} for
} if
}{ tx0 tfw sub ty0 moveto (#) show} ifelse
end
} def

/MatPrn % Print matrix Mnm formatted
% Call:      txtxt ytxt n m (name) Mnm MatPrn
% Requires: font, fh (fontheight), tms (mantissadigits), tmi (leading digits)
{1 dict
begin
  /Mnm exch def
  /nam exch def
  /m exch def
  /n exch def
  /y exch def
  /x exch def
  /a x def
  /fw fh 0.75 mul def
  /dx tms tmi add fw mul def
  /dy fh def
  x tmi 1 sub fw mul sub y moveto nam show
  /y y dy sub def
  /j 0 def
  n
  {/x a def
    m
    { x y Mnm j get Shownum
      /j j 1 add def
      /x x dx add def
    } repeat
  /y y dy sub def
} repeat
end
} def % MatPrn

```

```

/LF % n+2 linefeeds
% Call:      n LF
% Requires:  ytxt, fh(fontheight)
{ 2 add {/ytxt ytxt fh sub def} repeat
} def % LF

```

```

/GF % n+2 linefeeds, graphic feed
% Call:      n GF
% Requires:  ygrf, fh(fontheight)
{ 2 add {/ygrf ygrf fh sub def} repeat
} def % GF

```

```

/Bbox % Bounding box A4 portrait
% Call: Bbox
{/bx 595 def
/by 842 def
0 0 0 1 setcmykcolor
1 setlinewidth
0 0 moveto bx 0 lineto bx by lineto
0 by lineto closepath stroke
} def % Bbox

```

```

/Circle
% Call: x y r Circle
% Uses: global Lw setlinewidth
{1 dict
begin
newpath
/r exch def /y exch def /x exch def
x y r 0 360 arc
gsave
0 0 0 0 setcmykcolor
fill
grestore
% color external
stroke
end
} def % Circle

```

```

/ArrowTo
% Call: x y ArrowTo
% Uses: global Lw setlinewidth
% To be applied like lineto
% but contains already stroke and fill
{1 dict
begin
/y2 exch def
/x2 exch def
currentpoint
/y1 exch def
/x1 exch def
/a Lw 12 mul def % length of arrow
/b Lw 3 mul def % half width
/dx x2 x1 sub def
/dy y2 y1 sub def
/le dx dup mul dy dup mul add sqrt def
/an dy dx atan def

```

```

gsave
x1 y1 translate
an rotate
0 0 moveto le a sub 0 lineto stroke
newpath
le 0 moveto a neg b rlineto 0 b b add neg
rlineto closepath
fill
grestore
x2 y2 moveto
end
} def % ArrowTo

```

```

/VecFil % Fill vector by scalars
% Call: x1 x2 ... xn n vn VecFil
% Requires: none
% vn n array def
{1 dict
begin
/vn exch def
/n exch def
/i n def
1 1 n
{/i i 1 sub def
pop
/xi exch def
vn i xi put
} for
end
} def % VecFil

```

```

/MatFil % --> Anm % Identity matrix, square part
% Call: n m Mnm MatFil
% Requires: none
{1 dict
begin
/Anm exch def
/m exch def
/n exch def
/j 0 def
n m mul
{ Anm j 0 put
/j j 1 add def
} repeat
/j 0 def
n m gt {/n m def} if
n
{ Anm j 1 put
/j j m add 1 add def
} repeat
end
} def % MatFil

```

```

/MatSca % S1*Anm --> Bnm % Multiply by scalar
% Call: n m S1 Anm Bnm MatSca
% Requires: none
% S1 1 array def % for printing my MatPrn

```

```

{1 dict
begin
/Bnm exch def
/Anm exch def
/S1 exch def
/m exch def
/n exch def
/f S1 0 get def
/j 0 def
n m mul
{Bnm j Anm j get f mul put
/j j 1 add def
} repeat
end
} def % MatSca

/MatAdd % Anm+Bnm --> Cnm % Sum
% Call:      n m Anm Bnm Cnm MatAdd
% Requires: none
{1 dict
begin
/Cnm exch def
/Bnm exch def
/Anm exch def
/m exch def
/n exch def
/j 0 def
n m mul
{Cnm j Anm j get Bnm j get add put
/j j 1 add def
} repeat
end
} def % MatAdd

/MatSub % Anm-Bnm --> Cnm % Difference
% Call:      n m Anm Bnm Cnm MatAdd
% Requires: none
{1 dict
begin
/Cnm exch def
/Bnm exch def
/Anm exch def
/m exch def
/n exch def
/j 0 def
n m mul
{Cnm j Anm j get Bnm j get sub put
/j j 1 add def
} repeat
end
} def % MatSub

/MatMul % Anp*Bpm --> Cnm % Product
% Call:      n p m Anp Bpm Cnm MatMul
% Requires: none
{1 dict
begin

```

```

/Cnm exch def
/Bpm exch def
/Anp exch def
/m exch def
/p exch def
/n exch def
/ik 0 def
/i 0 def
n
{/k 0 def
m
{/s 0 def
/j 0 def
p
{/s s Anp i p mul j add get
      Bpm j m mul k add get mul add def
/j j 1 add def
} repeat % p
Cnm ik s put
/ik ik 1 add def
/k k 1 add def
} repeat % k
/i i 1 add def
} repeat % i
end
} def % MatMul

```

**/MatTrn** % Anm --> Bmn % Transposed

% Call: n m Anm Bmn MatTrn

% Requires: none

```

{1 dict
begin
/Bmn exch def
/Anm exch def
/m exch def
/n exch def
/i 0 def
m
{/k 0 def
n
{Bmn i n mul k add Anm k m mul i add get put
/k k 1 add def
} repeat
/i i 1 add def
} repeat
end
} def % MatTrn

```

**/MatEulC** % Euler angles --> R41

% Call: alf bet gam C41 MatEulC

% Requires: none

% Rotation matrix for camera angles

%  $X_4 = C_{41} * X_1$

```

{1 dict
begin
/C41 exch def
/gam exch def

```

```

/bet  exch  def
/alf  exch  def
/C31  9  array  def
/calf  alf  cos  def  /salf  alf  sin  def  /nalf  salf  neg  def
/cbet  bet  cos  def  /sbet  bet  sin  def  /nbet  sbet  neg  def
/cgam  gam  cos  def  /sgam  gam  sin  def  /ngam  sgam  neg  def
/Y21  [  cbet  0  nbet
        0    1  0
        sbet  0  cbet]  def
/X32  [  1  0  0
        0  calf  salf
        0  nalf  calf]  def
/Z43  [  cgam  sgam  0
        ngam  cgam  0
        0    0    1 ]  def
3 3 3  X32  Y21  C31  MatMul
3 3 3  Z43  C31  C41  MatMul
end
}  def  %  MatEulC

```

```

/MatLin  %  Ann*Xn=Yn --> Xn and det(Ann)
%  Call:      n  n  Ann  Xn  Yn  D1  MatLin
%  Requires:  none
%  First n is ignored
%  D1 1 array def
%  Recover Ann and Yn
%  Schwarz, Numerische Mathematik, Teubner / 1993
{1 dict
begin
/d1  exch  def
/Yn  exch  def
/Xn  exch  def
/Ann exch  def
/n   exch  def
  pop
/Yc  n  array  def
/Acc n  dup  mul  array  def
  Yn  Yc  copy
  Ann Acc  copy
%  Memory map; i,k on stack; top=N*i+k
/mp
{  exch
  n  mul  add
}  bind  def
%
/pa  n  array  def
/ca  n  array  def
/n1  n  1  sub  def
/dA  1  def
0 1  n1  1  sub
{/k  exch  def
/max  0  def
  pa  k  0  put
  k  1  n1
  {/i  exch  def
  /s  0  def
  k  1  n1

```

```

{/j exch def
/s s Ann i j mp get abs add def
} for %j
/bik Ann i k mp get abs def
/q bik s div def
q max gt {/max q def pa k i put } if
} for %i
pa k get k ne
{/dA dA neg def
0 1 n1
{/j exch def
/h Ann k j mp get def
Ann k j mp Ann pa k get j mp get put
Ann pa k get j mp h put
} for %j
} if
/dA dA Ann k k mp get mul def
k 1 add 1 n1
{/i exch def
/aik Ann i k mp get def
/akk Ann k k mp get def
/bkk akk abs def
bkk 1 lt
{/bik aik abs def
} if
Ann i k mp aik akk div put
k 1 add 1 n1
{/j exch def
Ann i j mp Ann i j mp get Ann i k mp get
Ann k j mp get mul sub put
} for %j
} for %i
} for %k
/dA dA Ann n1 n1 mp get mul def
0 1 n1 1 sub
{/k exch def
pa k get k ne
{/h Yn k get def
Yn k Yn pa k get get put
Yn pa k get h put
} if
} for %k
0 1 n1
{/i exch def
ca i Yn i get put
0 1 i 1 sub
{/j exch def
ca i ca i get Ann i j mp get ca j get mul sub put
} for %j
} for %i
n1 -1 0
{/i exch def
/s ca i get def
i 1 add 1 n1
{/k exch def
/s s Ann i k mp get Yn k get mul sub def
} for %k

```

```

/bik Ann i i mp get abs def
  Yn i s Ann i i mp get div put
} for %i
Yn Xn copy
d1 0 dA put % Determinant is an array with one element
Yc Yn copy % recovery
Acc Ann copy
end
} bind def % MatLin

```

### **/DrawGraph**

```

{grestore gsave SetHelv
/Lw 0.3 def
  Lw setlinewidth
+1 mm +1 mm scale
  xgrf1 ygrf1 translate
-1 1 scale
  0 0 0 1 setcmykcolor
  0 0 moveto ax 0 ArrowTo
  0 0 moveto 0 ax ArrowTo
  0 0 moveto cx cz ArrowTo
  0 0 r2 Circle
  cx cz translate
  0 0 moveto 0 ax lineto stroke
  0 0 moveto ax 0 lineto stroke
bet neg rotate
  0 0 moveto ax 0 ArrowTo
  0 0 moveto 0 ax ArrowTo
  0 0 moveto bu bw ArrowTo
  0 0 r2 Circle
  0 0 ax 0.8 mul 0 bet arc stroke
bu bw translate
  0 0 r2 Circle
  0 0 moveto ax 0 lineto stroke
  0 0 moveto 0 ax lineto stroke
psi neg rotate
  0 0 moveto ax 0 ArrowTo
  0 0 moveto 0 ax ArrowTo
  0 0 r2 Circle
  0 0 ax 0.8 mul 0 psi arc stroke
} def % DrawGraph

```

### **/DrawDots**

```

{grestore gsave SetHelv
/Lw 0.3 def
  Lw setlinewidth
+1 mm +1 mm scale
  xgrf ygrf translate
-1 1 scale
  0 0 0 1 setcmykcolor
  ax neg 0 moveto ax 0 lineto stroke
  0 ax neg moveto 0 ax lineto stroke
coll
  f1 g1 moveto f2 g2 lineto f3 g3 lineto f4 g4 lineto
closepath stroke
  f1 g1 r1 Circle
  f2 g2 r1 Circle

```



```

f3 g3 r1 Circle
f4 g4 r1 Circle
col2
f1c g1c r1 Circle
f2c g2c r1 Circle
f3c g3c r1 Circle
f4c g4c r1 Circle
} def % DrawDots

```

### **/Initial**

```

{
/cc 3 array def
/bc 3 array def
/ca 3 array def
/ba 3 array def
/bcc 3 array def
/bac 3 array def

/n 8 def
/m 6 def
/M86 n m mul array def
/M68 m n mul array def
/N66 m m mul array def
/r81 n array def
/n61 m array def
/p61 m array def
/D1 1 array def
/y81 n array def
/z81 n array def
/y61 n array def
/z61 n array def

/C41 9 array def /C33 9 array def
/A41 9 array def /B33 9 array def /B33c 9 array def
/k.1 3 array def /k.2 3 array def /k.3 3 array def /k.4 3 array def
/u.1 3 array def /u.2 3 array def /u.3 3 array def /u.4 3 array def

/ane apo neg def
/bne bpo neg def
apo bpo 0 3 k.1 VecFil
ane bpo 0 3 k.2 VecFil
ane bne 0 3 k.3 VecFil
apo bne 0 3 k.4 VecFil

alf bet gam 3 ca VecFil
cx cy cz 3 cc VecFil

/xtxt txt1 def
/ytxt ytxt1 def
/xgrf xgrf2 def
/ygrf ygrf2 def

gsave

} def % Initial

```

## /Mapping

```
{psi the phi 3 ba VecFil  
bu bv bw 3 bc VecFil
```

```
alf bet gam C41 MatEulC  
the psi phi A41 MatEulC
```

```
3 3 C41 C33 MatTrn  
3 3 A41 B33 MatTrn
```

```
3 3 1 B33 k.1 u.1 MatMul  
3 3 1 B33 k.2 u.2 MatMul  
3 3 1 B33 k.3 u.3 MatMul  
3 3 1 B33 k.4 u.4 MatMul
```

```
3 1 u.1 bc u.1 MatAdd  
3 1 u.2 bc u.2 MatAdd  
3 1 u.3 bc u.3 MatAdd  
3 1 u.4 bc u.4 MatAdd
```

```
/f1q u.1 0 get u.1 2 get div def  
/f2q u.2 0 get u.2 2 get div def  
/f3q u.3 0 get u.3 2 get div def  
/f4q u.4 0 get u.4 2 get div def  
/g1q u.1 1 get u.1 2 get div def  
/g2q u.2 1 get u.2 2 get div def  
/g3q u.3 1 get u.3 2 get div def  
/g4q u.4 1 get u.4 2 get div def
```

```
/f1 f1q d mul def  
/f2 f2q d mul def  
/f3 f3q d mul def  
/f4 f4q d mul def  
/g1 g1q d mul def  
/g2 g2q d mul def  
/g3 g3q d mul def  
/g4 g4q d mul def
```

```
% Scaling for numerical improvement
```

```
/Sc Sc1 0 get def  
/apos apo Sc mul def  
/anes ane Sc mul def  
/bpos bpo Sc mul def  
/bnes bne Sc mul def
```

```
/M86 [-1 0 f1q anes f1q mul bpos f1q mul bpos  
-1 0 f2q apos f2q mul bpos f2q mul bpos  
-1 0 f3q apos f3q mul bnes f3q mul bnes  
-1 0 f4q anes f4q mul bnes f4q mul bnes  
0 -1 g1q anes g1q mul bpos g1q mul anes  
0 -1 g2q apos g2q mul bpos g2q mul apos  
0 -1 g3q apos g3q mul bnes g3q mul apos  
0 -1 g4q anes g4q mul bnes g4q mul anes ] def
```

```

/r81 [ apos
      anes
      anes
      apos
      bpos
      bpos
      bnes
      bnes ] def

8 6      M86 M68 MatTrn
6 8 6 M68 M86 N66 MatMul
6 8 1 M68 r81 n61 MatMul

6 6 N66 p61 n61 D1 MatLin

/buc p61 0 get def
/bvc p61 1 get def
/bwc p61 2 get def
/psc p61 3 get gr def % rad to grad
/thc p61 4 get gr def
/phc p61 5 get gr def

% Rescaling
/Sc Sc1 0 get
/buc buc Sc div def
/bvc bvc Sc div def
/bwc bwc Sc div def

buc bvc bwc 3 bcc VecFil
psc thc phc 3 bac VecFil
} def % Mapping

/Reverse
{thc psc phc A41 MatEulC
 3 3 A41 B33c MatTrn
 3 3 1 B33c k.1 u.1 MatMul
 3 3 1 B33c k.2 u.2 MatMul
 3 3 1 B33c k.3 u.3 MatMul
 3 3 1 B33c k.4 u.4 MatMul
 3 1 u.1 bc u.1 MatAdd
 3 1 u.2 bc u.2 MatAdd
 3 1 u.3 bc u.3 MatAdd
 3 1 u.4 bc u.4 MatAdd
/f1c u.1 0 get u.1 2 get div d mul def
/f2c u.2 0 get u.2 2 get div d mul def
/f3c u.3 0 get u.3 2 get div d mul def
/f4c u.4 0 get u.4 2 get div d mul def
/g1c u.1 1 get u.1 2 get div d mul def
/g2c u.2 1 get u.2 2 get div d mul def
/g3c u.3 1 get u.3 2 get div d mul def
/g4c u.4 1 get u.4 2 get div d mul def
} def % Reverse

```

### **/Params**

```
{grestore gsave SetHelv
+1 mm +1 mm scale
0 0 0 1 setcmykcolor
txt1 ytxt 3 1 (alf bet gam) ca MatPrn 3 LF
txt1 ytxt 3 1 (cx cy cz ) cc MatPrn 3 LF
/dd [d] def
txt1 ytxt 1 1 (d) dd MatPrn 1 LF
/ab [apo bpo] def
txt1 ytxt 2 1 (a b) ab MatPrn
txt2 ytxt 1 1 (Sc) Sc1 MatPrn 2 LF
} def % Params
```

### **/Result**

```
{grestore gsave SetHelv
+1 mm +1 mm scale
0 0 0 1 setcmykcolor
0.3 setlinewidth
/ytxt0 ytxt def
col1
txt1 11 sub ytxt 2.1 add 0.3 -33 rectfill
0 0 0 1 setcmykcolor
txt1 ytxt 3 1 (psi the phi (true)) ba MatPrn 3 LF
0 0 0 1 setcmykcolor
txt1 ytxt 3 1 (bu bv bw (true)) bc MatPrn 3 LF
/ytxt ytxt0 def
col2
newpath txt2 11 sub ytxt 1 add 1 0 360 arc stroke
0 0 0 1 setcmykcolor
txt2 ytxt 3 1 (psi the phi (calc)) bac MatPrn 3 LF
0 0 0 1 setcmykcolor
txt2 ytxt 3 1 (bu bv bw (calc)) bcc MatPrn
col2
newpath txt2 11 sub ytxt 1 add 1 0 360 arc stroke 3 LF
} def % Result
```

### **/Checks**

```
{grestore gsave SetHelv
+1 mm +1 mm scale
0 0 0 1 setcmykcolor
txt1 ytxt 8 6 (M86) M86 MatPrn
txt3 ytxt 8 1 (r81) r81 MatPrn 8 LF
txt1 ytxt 6 6 (N66) N66 MatPrn
txt3 ytxt 6 1 (n61) n61 MatPrn 6 LF
8 6 1 M86 p61 y81 MatMul
8 1 y81 r81 z81 MatSub
txt1 ytxt 8 1 (resM) z81 MatPrn
6 6 1 N66 p61 y61 MatMul
6 1 y61 n61 z61 MatSub
/tms 10 def
txt2 ytxt 6 1 (resN) z61 MatPrn 5 LF
txt2 ytxt 1 1 (Det N66) D1 MatPrn 1 LF
} def % Checks
```

### **/Make1**

```
{Mapping Result Reverse DrawDots 5 LF 15 GF Text2
} def % Make1
```

### **/Text1**

```
{grestore gsave SetHelv
 0 0 0 1 setcmykcolor
 1 mm 1 mm scale
/fh 11 mi def
/Helvetica-Bold findfont fh scalefont setfont
145 19 moveto (x) show
184 19 moveto (y) show
179 51 moveto (z) show
131 53 moveto (u) show
168 59 moveto (v) show
158 91 moveto (w) show
 53 213 moveto (k) show
 88 229 moveto (l) show
 69 257 moveto (m) show
/Symbol findfont fh scalefont setfont
135 56 moveto (b) show
 56 219 moveto (y) show
} def
```

### **/Text2**

```
{grestore gsave SetHelv
 1 mm 1 mm scale
/fh 11 mi def
/Helvetica-Bold findfont fh scalefont setfont
116 239 moveto (f) show
148.5 272 moveto (g) show
} def
```

### **/Page0**

```
{%Bbox
/psi 15 def /the 0 def /phi 0 def
/bu 50 def /bv 0 def /bw 180 def
Initial
Params DrawGraph Mapping Result Checks Reverse DrawDots
Text1 Text2
} def % Page0
```

### **/Page1**

```
{Initial
/psi 0 def /the 0 def /phi 0 def
/bu 0 def /bv 0 def /bw 180 def
Make1
/psi 15 def /the 0 def /phi 0 def
/bu 0 def /bv 0 def /bw 180 def
Make1
/psi 0 def /the 15 def /phi 0 def
/bu 0 def /bv 0 def /bw 180 def
Make1
/psi 0 def /the 0 def /phi 15 def
/bu 0 def /bv 0 def /bw 180 def
Make1
} def % Page1
```

## **/Page2**

```
{Initial
/psi 15 def /the 15 def /phi 0 def
/bu 0 def /bv 0 def /bw 180 def
Make1
/psi 15 def /the 0 def /phi 15 def
/bu 0 def /bv 0 def /bw 180 def
Make1
/psi 0 def /the 15 def /phi 15 def
/bu 0 def /bv 0 def /bw 180 def
Make1
/psi 15 def /the 15 def /phi 15 def
/bu 0 def /bv 0 def /bw 180 def
Make1
} def % Page2
```

## **/Page3**

```
{Initial
/psi 0 def /the 0 def /phi 0 def
/bu 0 def /bv 0 def /bw 180 def
Make1
/psi 0 def /the 0 def /phi 0 def
/bu 100 def /bv 0 def /bw 180 def
Make1
/psi 0 def /the 0 def /phi 0 def
/bu 0 def /bv 100 def /bw 180 def
Make1
/psi 0 def /the 0 def /phi 0 def
/bu 100 def /bv 100 def /bw 180 def
Make1
} def % Page3
```

% Choose one

```
/sel 0 def
```

```
sel 0 eq {Page0} if
sel 1 eq {Page1} if
sel 2 eq {Page2} if
sel 3 eq {Page3} if
```

```
showpage
```

## 12. References

- [1] H.R.Schwarz  
Numerische Mathematik, B.G.Teubner, Stuttgart, 1993
- [2] R.C.Hughes  
Enhanced single camera photogrammetry algorithms for real time control applications  
[http://www.isprs.org/proceedings/XXVI/part5/391\\_XXVI-part5.pdf](http://www.isprs.org/proceedings/XXVI/part5/391_XXVI-part5.pdf)
- [3] PostScript interpreter, editor and debugger PSAlter  
<http://www.quite.com/psalter/>
- [4] Teaching a robot by showing the motion  
<http://docs-hoffmann.de/icip01.pdf>
- [5] Euler angles, rotation matrices and projections  
<http://docs-hoffmann.de/euler26112001.pdf>
- [6] Planar projections  
<http://docs-hoffmann.de/project18032004.pdf>
- [7] Illustrations for a lecture Computer Vision  
<http://docs-hoffmann.de/covi-2005-1-4.pdf>
- [8] PostScript Tutor with references  
<http://docs-hoffmann.de/pstutor22112002.pdf>
- [9] All documents by G.Hoffmann  
<http://docs-hoffmann.de/howww41a.html>

This document:

<http://docs-hoffmann.de/track1cam15062012.pdf>

Gernot Hoffmann

July 01 / 2012 - February 06 / 2013

Website

Load Browser / Click here