Camera Calibration

Lab Exercise 3 for the Computer Vision Course

Paul Withagen & Rein van den Boomgaard
University of Amsterdam,
The Netherlands

October 28, 2002

Abstract

This lab exercise investigates the algorithm to calibrate a camera. The calculated camera parameters will be used to draw objects in the scene. We follow the book [Trucco98], chapter 6.

Together with exercises two and four we will work towards the measurement of an object in a real-world scene for which images from different locations are available.

In chapter 1 and 2 we give some theory and code to do camera calibration and to calculate the camera parameters. Then in chapter 3 the problems we want you to solve are given. In chapter 4 we explain what we expect from you and in Appendix A some comments are given about the different use of T in chapter 2 and 6 of [Trucco98].

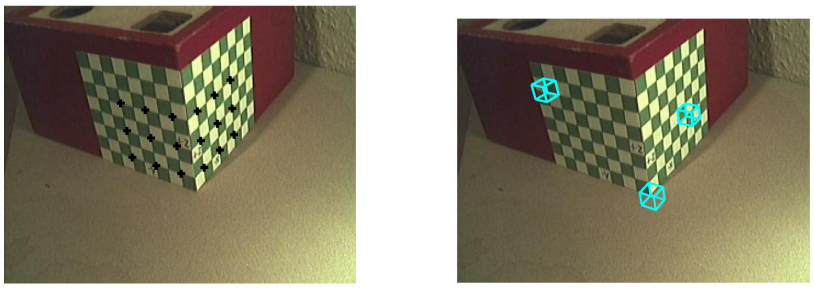

Figure 1: An image of a calibration rig, left with calibration points, right with cubes to check the calibration.

1 Estimating the camera projection matrix

In this section we consider the camera projection to be a black box. All we assume here is the relation:

$$\begin{pmatrix} \lambda_i x_i \\ \lambda_i y_i \\ \lambda_i \end{pmatrix} = M \begin{pmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{pmatrix}$$

between a point in 3D space (X_i, Y_i, Z_i) and the depicted point on the retina (x_i, y_i). This relationship (the matrix M) can be learned from examples.

Let the points (x_i, y_i) for $i = 1, \ldots, n$ be collected in a Matlab array `xy` such that `xy(i,1)` equals x_i and `xy(i,2)` equals y_i, i.e. the i-th row in the matrix `xy` is the point (x_i, y_i). Equivalently let the corresponding 3D points (X_i, Y_i, Z_i) be collected in the matrix `XYZ`.

Figure 2: **Calibration object.** The calibration object consists of two perpendicular planes. On the planes square tiles are depicted (the sides of the tiles are 1cm). A convenient world coordinate frame is graphically overlayed on the image. The retina coordinates together with the corresponding 3D coordinates of the points indicated with a '+' are available in the file `CV3_calibrationpoints.mat`.

In Fig. 2 an image of a calibration object is shown. The 18 points marked with a '+' are available in the file `CV3_calibrationpoints.mat`. Loading this file (`load CV3_calibrationpoints`) will set the `xy` and `XYZ` matrix. An interactive program to collect points in 2D and corresponding points in is given in `CV3_collectPoints.m`.

The camera projection matrix M can be estimated by solving the homoge-

neous linear system (see Eq. 6.17 in [Trucco98]):

$$A\mathbf{m} = \mathbf{0} \tag{1}$$

with

$$A = \begin{pmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 & -x_1Z_1 & -x_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -y_1X_1 & -y_1Y_1 & -y_1Z_1 & -y_1 \\ X_2 & Y_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & -x_2X_2 & -x_2Y_2 & -x_2Z_2 & -x_2 \\ 0 & 0 & 0 & 0 & X_2 & Y_2 & Z_2 & 1 & -y_2X_2 & -y_2Y_2 & -y_2Z_2 & -y_2 \\ . & . & . & . & . & . & . & . & . & . & . & . \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -x_nX_n & -x_nY_n & -x_nZ_n & -x_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -y_nX_n & -y_nY_n & -y_nZ_n & -y_n \end{pmatrix} \tag{2}$$

and

$$\mathbf{m} = \begin{pmatrix} m_{11}, m_{12}, \ldots, m_{33}, m_{34} \end{pmatrix}^{\mathsf{T}}. \tag{3}$$

Given the matrices `xy` and `XYZ` it is relatively easy to construct the matrix `A`:

3a ⟨*Construct A-matrix* 3a⟩≡

```
x = xy(:,1);
y = xy(:,2);
X = XYZ(:,1);
Y = XYZ(:,2);
Z = XYZ(:,3);
o = ones( size(x) );
z = zeros( size(x) );
Aoddrows  = [ X Y Z o z z z z -x.*X -x.*Y -x.*Z -x ];
Aevenrows = [ z z z z X Y Z o -y.*X -y.*Y -y.*Z -y ];
A = [Aoddrows; Aevenrows];
```

This code is used in chunk 4b.

It should be noted that the matrix `A` in the above code fragment is *not* the matrix as defined in equation Eq. 2. The solution of the homogeneous system is not changed by taking the `A` matrix defined in the above Matlab code instead (why not?).

The non-trivial solution of the homogeneous equation $A\mathbf{m} = \mathbf{0}$ is found by selecting for $\mathbf{m}$ the last column of the matrix V that results from the singular value decomposition of the matrix A. The Matlab code is:

3b ⟨*Calculate solution of A m = 0* 3b⟩≡

```
[U, D, V] = svd( A );
m = V(:,end);
```

This code is used in chunk 4b.

Reshaping `m` back into a 3×4 matrix is a bit tricky. The Matlab convention for 'linearizing' matrices is to put the elements *column wise* into a large vector. The `m` vector is obtained from `M` by putting the elements *row wise* into a large vector. The solution is to reshape `m` into a 4×3 matrix whose transpose is the camera projection matrix `M`:

4a ⟨*Reshape m vector into M matrix* 4a⟩≡

```
M = reshape(m,4,3)';
```

This code is used in chunk 4b.

The presented code fragments can be put together in the function `CV3_estimateProjectionMatrix` to estimate `M` given the calibration points (collected in matrices `xy` and `XYZ`).

4b ⟨*CV3_estimateProjectionMatrix.m* 4b⟩≡

```
function M = CV3_estimateProjectionMatrix( xy, XYZ )
⟨Construct A-matrix 3a⟩
⟨Calculate solution of A m = 0 3b⟩
⟨Reshape m vector into M matrix 4a⟩
```

Root chunk (not used in this document).

The following Matlab code fragment loads the calibration points and then estimates the projection matrix

4c ⟨*EstimateMatrix* 4c⟩≡

```
load CV3_calibrationpoints
M = CV3_estimateProjectionMatrix( xy, XYZ );
```

Root chunk (not used in this document).

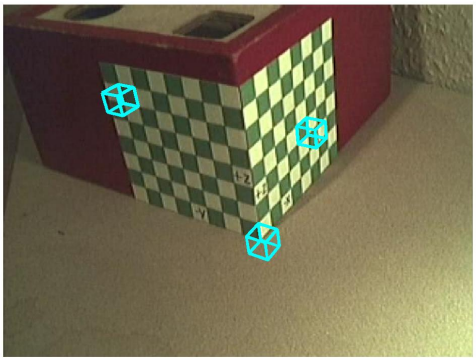

Figure 3: **Drawing 3D cubes in the scene.** The calibrated camera enables us to draw 3D objects in the scene.

The projection matrix estimated in this section and the knowledge from exercise 2 can be used to render 3D figures in the scene depicted in Fig. 2. Here we take a simple cube with sides of 1cm. These corresponds with the squares on the calibration pattern. This allows for an easy visual check of the projection matrix estimation.

2 Camera parameters from the projection matrix

In practical situations it is not always sufficient to have estimated the projection matrix M, The camera parameters are needed. In this section we assume that the projection matrix has been estimated with the procedure from the previous sections. The estimated projection matrix is denoted as $\hat{M}$:

$$\hat{M} = \begin{pmatrix} \hat{m}_{11} & \hat{m}_{12} & \hat{m}_{13} & \hat{m}_{14} \\ \hat{m}_{21} & \hat{m}_{22} & \hat{m}_{23} & \hat{m}_{24} \\ \hat{m}_{31} & \hat{m}_{32} & \hat{m}_{33} & \hat{m}_{34} \end{pmatrix}.$$

For easy of reference later on we will subdivide the matrix $\hat{M}$ into 4 3-element vectors:

$$\hat{M} = \begin{pmatrix} \mathbf{q}_1^\top & \\ \mathbf{q}_2^\top & \mathbf{q}_4 \\ \mathbf{q}_3^\top & \end{pmatrix}$$

where $\mathbf{q}_i$ is a 3 element vector.

It should be noted that as always when working with homogeneous representation of vectors and operators in space the test on equality should be read as

equality up to a scale factor. In this case we would like to find the camera parameters f_x, f_y, o_x, o_y, the rotation matrix R (with elements r_{ij}) and the translation vector with elements T_x, T_y and T_z. We thus are looking for those parameters such that:

$$\hat{M} = \gamma M$$

where

$$M = \begin{pmatrix} -f_x r_{11} + o_x r_{31} & -f_x r_{12} + o_x r_{32} & -f_x r_{13} + o_x r_{33} & -f_x T_x + o_x T_z \\ -f_y r_{21} + o_y r_{31} & -f_y r_{22} + o_y r_{32} & -f_y r_{23} + o_y r_{33} & -f_y T_y + o_y T_z \\ r_{31} & r_{32} & r_{33} & T_z \end{pmatrix}. \tag{4}$$

Note that the rows and columns of a rotation matrix form an orthonormal basis. This leads to:

$$|\gamma| = \sqrt{\mathbf{q}_3^\mathsf{T}\mathbf{q}_3}.$$

We now set $\sigma = \gamma/|\gamma|$ and $\hat{M} = \hat{M}/|\gamma|$ and obtain:

$$\begin{aligned} T_z &= \sigma\hat{m}_{34} && (5)\\ r_{3i} &= \sigma\hat{m}_{3i}, \qquad i = 1,2,3 && (6)\\ o_x &= \mathbf{q}_1^\mathsf{T}\mathbf{q}_3 && (7)\\ o_y &= \mathbf{q}_2^\mathsf{T}\mathbf{q}_3 && (8)\\ f_x &= \sqrt{\mathbf{q}_1^\mathsf{T}\mathbf{q}_1 - o_x^2} && (9)\\ f_y &= \sqrt{\mathbf{q}_2^\mathsf{T}\mathbf{q}_2 - o_y^2} && (10)\\ r_{1i} &= \sigma(o_x\hat{m}_{3i} - \hat{m}_{1i})/f_x && (11)\\ r_{2i} &= \sigma(o_y\hat{m}_{3i} - \hat{m}_{2i})/f_y && (12)\\ T_x &= \sigma(o_x\hat{m}_{34} - \hat{m}_{14})/f_x && (13)\\ T_y &= \sigma(o_y\hat{m}_{34} - \hat{m}_{24})/f_y && (14) \end{aligned}$$

It should be noted that there is an error in [Trucco98] on page 135. The above equations for T_x and T_y differ slightly from their definition.

From Eq. 5 we see that σ is equal to the sign of $\hat{m}_{34}$. Thus in case the origin of the world frame is in front of the camera we take $\sigma = \hat{m}_{34}/|\hat{m}_{34}|$ else we set $\sigma = -\hat{m}_{34}/|\hat{m}_{34}|$.

The rotation matrix $\hat{R}$ obtained with this estimation procedure is not guaranteed to be orthogonal. Therefore we calculate the rotation matrix that is closest to the estimated matrix (in the Frobenius norm sense). Let $\hat{R} = UDV^\mathsf{T}$ then $R = UV^\mathsf{T}$.

The above equations can be easily 'translated' into Matlab. We do so without comments.

6 ⟨*CV3_estimateCameraParameters.m* 6⟩≡

```
function [fx, fy, ox, oy, R, T] = ...
 CV3_estimateCameraParameters( M, inFront )
    % estimate camera parameters given projection matrix
    M = M / sqrt(M(3,1)^2+M(3,2)^2+M(3,3)^2);
    if inFront
        s = sign(M(3,4));
    else
        s = -sign(M(3,4));
    end
    T(3) = s*M(3,4);
    R = zeros(3,3);
    R(3,:)=s*M(3,1:3);
    q1 = M(1,1:3)';
    q2 = M(2,1:3)';
    q3 = M(3,1:3)';
    q4 = M(1:3,4);
    ox = q1'*q3;
    oy = q2'*q3;
    fx = sqrt( q1'*q1 - ox^2 );
    fy = sqrt( q2'*q2 - oy^2 );
    R(1,:) = s*(ox*M(3,1:3) - M(1,1:3) ) / fx;
    R(2,:) = s*(oy*M(3,1:3) - M(2,1:3) ) / fy;
    T(1) = s*(ox*M(3,4) - M(1,4) ) / fx;
    T(2) = s*(oy*M(3,4) - M(2,4) ) / fy;
    T = T';
    [U,D,V] = svd(R);
    R = U*V';
```

Root chunk (not used in this document).

To check the above computations we can use Eq. 4 to construct M from the parameters.

7 ⟨*CV3_projectionMatrixFromParameters.m* 7⟩≡

```
function M = CV3_projectionMatrixFromParameter( fx, fy,
                                                ox, oy, R, T )
% construct projection matrix from parameters
M = [
 -fx*R(1,1)+ox*R(3,1), -fx*R(1,2)+ox*R(3,2), ...
 -fx*R(1,3)+ox*R(3,3), -fx*T(1)+ox*T(3);
 -fy*R(2,1)+oy*R(3,1), -fy*R(2,2)+oy*R(3,2), ...
 -fy*R(2,3)+oy*R(3,3), -fy*T(2)+oy*T(3);
 R(3,1),         R(3,2),          R(3,3),           T(3)
];
```

Root chunk (not used in this document).

This reconstructed matrix can be used to draw the cubes in the image. Compare the results with the drawings that were made in a previous section.

3 Exercises

3.1 Accuracy I

Redo the camera calibration as introduced in the previous sections but now using more calibration points (a rule of the thumb is that accurate results need about 20 to 30 calibration points). Report the calculated camera parameters.

3.2 Accuracy II

Figure 4: **Different views of the calibration rig.** The four views are depicted in images CV3_view1.jpg, CV3_view2.jpg, CV3_view3.jpg, and CV3_view4.jpg.

Redo the calibration but now for different views of the calibration rig. Obviously the internal camera parameters should in theory be constant. The different views are given in Fig. 4 and are available in the following tif files: `CV3_view1.jpg, CV3_view2.jpg, CV3_view3.jpg, and CV3_view4.jpg`.

You can use these points and the calculated camera parameters in the Stereo Vision exercise (Exercise 4).

3.3 Direct Estimation of Camera Parameters

In section 6.2 of [Trucco98] a calibration procedure is described that directly estimates the internal and external camera parameters (Algorithm EXPL_PARS_CAL).

Write the Matlab code to implement the calibration procedure. Again you can interactively select your calibration points (or take the ones in the `CV3_calibrationpoints.mat` file.

The image center can be taken as the outcome of the calibration procedure explained and implemented in this report.

3.4 Example code

The code in this report is available as m-files and the figures in this exercise are generated using `CV3_calibration.m`.

Also given is the program `CV3_collectPoints.m`, which enables you to define the calibration points interactively. The program first gives the opportunity to zoom in on the image (all points need to be visible though). Enter `return` when you want to start collecting the points.Then the program asks for the XYZ coordinates of the 3D point (each coordinate separately) and then you have the opportunity to select a point in the image interactively. You have to select a 2D point by clicking on it.

Evidently a much more user friendly version of this program is needed that uses a GUI to present the user with a form to fill in with the coordinates. If there are any volunteers. . . .

4 The hand-in

A small report is expected which solves all problems in this document. Include figures, images, tables, and code where appropriate. You could also include a running version of your code, but the report should be sufficiently detailed to base your grade on. If you do send code, name the function we should run to recreate all your results `runme.m` and don't forget to use a lot of comments in your code. Explain what a function does, what problem you are solving with it, which variables you use etc..

A solution to most of the problems presented in a reasonable report will be rewarded with a six. For higher grades the report should be better, and for a nine or ten you should really do something extra. Extra credits can be achieved by for example trying the algorithms on your own dataset (from the Internet for example), writing a nice GUI around your software, improving the given algorithms, illustrating the theoretical answer by experiments you did, etc.

Bundle any code and the report (pdf preferred, MS Word acceptable) in a zip or tgz-file, and mail it to `compvision@science.uva.nl`. Do not forget to write you names and student numbers in the e-mail, report, and code. Do this before the deadline: **Monday 4 November, 23:59**.

See the practicum webpage:
`http://carol.wins.uva.nl/~paulw/CV2002Prac.html` for deadlines, updates, bugs etc. For questions you can email to
`compvision@science.uva.nl`, but check the website for FAQs and for info when help is available.

A The use of T in [Trucco98]

The use of T in [Trucco98] is not done very consequently. See the note on page 126. This means you can convert the translation matrices between chapters by:

$$T_6 = -RT_2,$$

where the indices refer to chapters in the book. Keep this in mind when you use code from the previous exercise.

Literature

[Trucco98] Emanuele Trucco and Alessandro Verri, “Introductory techniques of 3-D computer vision”, Prentice Hall, 1998